

WebSphere Application Server for z/OS V5.0.1:



Applications

Note

Before using this information, be sure to read the general information under “Notices” on page 585.

Compilation date: September 23, 2003

© Copyright International Business Machines Corporation 2003. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

How to send your comments ix

Chapter 1. Welcome to Applications . . . 1

Chapter 2. Using Web applications . . . 7

Web applications 7

web.xml file 8

Migrating Web application components 9

Default Application. 13

Servlets. 14

Developing servlets with WebSphere Application

Server extensions 14

 Application lifecycle listeners and events 15

 Listener classes for servlet context and session

 changes. 15

 Example:

 com.ibm.websphere.DBConnectionListener.java . 16

 Servlet filtering 16

 Filter, FilterChain, FilterConfig classes for servlet

 filtering. 17

 Example: com.ibm.websphere.LoggingFilter.java 17

 Configuring page list servlet client configurations 18

 autoRequestEncoding and autoResponseEncoding 21

 Examples: autoRequestEncoding and

 autoResponseEncoding encoding examples . . . 21

JavaServer Pages files 22

Developing JavaServer Pages files with WebSphere

extensions 22

 Tag libraries 23

 tsx:dbconnect tag JavaServer Pages syntax . . . 23

 dbquery tag JavaServer Pages syntax 25

 dbmodify tag JavaServer Pages syntax 26

 tsx:getProperty tag JavaServer Pages syntax and

 examples 26

 tsx:userid and tsx:passwd tag JavaServer Pages

 syntax 27

 tsx:repeat tag JavaServer Pages syntax 27

 Example: Combining tsx:repeat and

 tsx:getProperty JavaServer Pages tags. 28

 Example: tsx:dbmodify tag syntax 28

 Example: Using tsx:repeat JavaServer Pages tag

 to iterate over a results set 29

 JspBatchCompiler tool. 30

Bean Scripting Framework 32

Developing Web applications 32

 Disabling JavaServer Pages run-time compilation 33

Example: Converting JavaScript source to the Bean

Scripting Framework 34

Scenario: Creating a Bean Scripting Framework

application 35

Example: Bean Scripting Framework code example 40

Web modules. 44

Assembling Web Modules 44

 Context parameters. 46

 Security constraints. 46

Servlet mappings 46

Invoker attributes 46

Error pages 46

File serving 47

Initialization parameters 47

Servlet caching 47

Web components 47

Web property extensions 47

Web resource collections 47

Welcome files. 47

Context parameter assembly settings 47

Initialization parameter assembly settings . . . 48

Filter assembly settings 48

JavaServer Pages attribute assembly settings . . 49

Multipurpose Internet Mail Extensions (MIME)

filter assembly settings 51

Page list assembly settings 53

Security constraint assembly settings 53

Servlet mapping assembly settings 54

Tag library assembly settings 55

Welcome file assembly settings 55

Servlet caching configuration assembly settings 55

Web components assembly settings 57

Web modules assembly settings 59

Assembly property extensions 62

File serving attribute assembly settings 63

Invoker attribute assembly settings 63

Error page assembly settings 63

Web resource collections security constraint

properties 64

Troubleshooting tips for Web application

deployment 65

Modifying the default Web container configuration 66

 Web container 67

 Web container settings. 67

 Web module settings 67

 Web Module Deployment settings. 68

 Web container advanced settings 69

 Web container custom property settings 70

Web applications: Resources for learning 71

Chapter 3. Managing HTTP sessions 73

Sessions 73

Migrating HTTP sessions 73

Developing session management in servlets . . . 75

 Example: SessionSample.java 76

Assembling so that session data can be shared . 76

Session security support 77

Session management support 78

Configuring session management by level 79

Session tracking options 80

 Session tracking with cookies 80

 Session tracking with URL rewriting 80

 Session tracking with SSL information 81

Configuring session tracking 82

Serializing access to session data	82
Session Management settings	83
Cookie settings	86
Distributed sessions	87
Session recovery support	87
Distributed Environment settings	88
Configuring for database session persistence	88
Switching to a multirow schema	89
Creating a DB2 table for session persistence	89
Database settings	91
Multirow schema considerations	92
Memory to memory replication.	93
Memory to memory topology: Peer to peer with local replicator	94
Configuring for memory to memory replication	95
Memory-to-memory sessions settings.	95
Clustered session support	96
Tuning session management.	96
Configuring scheduled invalidation	97
Configuring write contents	97
Configuring write frequency.	98
Base in-memory session pool size	99
Controlling write operations.	99
Tuning parameter settings	100
Tuning parameter custom settings	102
Best practices for using HTTP Sessions	103
Managing HTTP sessions: Resources for learning:	107

Chapter 4. Using enterprise beans in applications 109

Enterprise beans	109
Developing enterprise beans	110
Migrating enterprise bean code to the supported specification	111
WebSphere extensions to the Enterprise JavaBeans specification	114
Best practices for developing enterprise beans	115
Using access intent policies	115
Access intent policies	115
Applying access intent policies to methods	117
Access intent exceptions	118
Access intent assembly settings	118
Access intent best practices.	120
Frequently asked questions: Access intent	120
EJB modules.	121
Assembling EJB modules	121
CMP field assembly settings	123
Container transactions	123
Container transaction assembly settings	123
EJB module assembly settings	125
Entity bean assembly settings	126
Message-driven bean assembly settings.	135
Method extensions	137
Method extension assembly settings	137
Method permissions	140
Method permission assembly settings	140
Query assembly settings.	141
References	141
EJB reference assembly settings	141
EJB local-reference assembly settings	142
EJB relation assembly settings	143

Exclude list assembly settings	143
Security role assembly settings	144
Session bean assembly properties.	145
EJB containers	149
Managing EJB containers	150
EJB container settings	150
EJB container system properties	151
EJB cache settings	151
Container interoperability	152
Deploying EJB modules	156
EJB module collection	156
EJB module settings	156
Troubleshooting tips for EJBDEPLOY relationships.	157
Enterprise beans: Resources for learning	157
EJB method Invocation Queuing	158

Chapter 5. Using message-driven beans in applications 161

Message-driven beans - an overview	161
Message-driven beans - components.	162
Message-driven beans - transaction support	164
Designing an enterprise application to use message-driven beans	164
Developing an enterprise application to use message-driven beans	166
Deploying an enterprise application to use message-driven beans	168
Configuring deployment attributes for a message-driven bean	168
Configuring message listener resources for message-driven beans	170
Configuring the message listener service	170
Adding a new listener port.	178
Configuring a listener port	178
Deleting a listener port	179
Configuring security for message-driven beans	179
Administering listener ports	180
Important files for message-driven beans and extended messaging	181
Troubleshooting message-driven beans	182
Message-driven beans samples	182

Chapter 6. Using application clients 185

Application clients	185
Application client functions	187
J2EE application clients	189
Pluggable application clients	189
Developing J2EE application client code	191
J2EE application client class loading.	195
Developing pluggable application client code.	197
Assembling Application Client Modules	198
Application client assembly settings	199
Deploying application clients on z/OS	200
Application Client Resource Configuration Scripting tool for z/OS	201
Determining required properties for z/OS application client resources.	203
Deploying application clients on workstation platforms.	211

Starting the Application Client Resource Configuration Tool and opening an EAR file	211
Data sources for application clients	211
Configuring new data source providers (JDBC providers) for application clients	212
Configuring new data sources for application clients	215
Configuring mail providers and sessions for application clients	215
Configuring new mail sessions for application clients	219
URLs for application clients	219
URL providers for the Application Client Resource Configuration Tool	219
Configuring new URL providers for application clients	220
Configuring new URLs with the Application Client Resource Configuration Tool	222
WebSphere asynchronous messaging using the Java Message Service API for the Application Client Resource Configuration Tool	223
Configuring Java messaging client resources	223
Configuring new connection factories for application clients	258
Configuring new Java Message Service destinations for application clients	259
Example: Configuring MQ Queue and Topic connection factories and destination factories for application clients	259
Example: Configuring WAS Queue and Topic connection factories and destination factories for application clients	261
Configuring new resource environment providers for application clients	262
Configuring new resource environment entries for application clients.	263
Managing application clients	264
Updating data source and data source provider configurations with the Application Client Resource Configuration Tool	265
Updating URLs and URL provider configurations for application clients	265
Updating mail session configurations for application clients	265
Updating Java Message Service provider, connection factories, and destination configurations for application clients	266
Updating MQ Java Message Service provider, MQ connection factories, and MQ destination configurations for application clients	266
Updating Resource Environment Entry and Resource Environment Provider configurations for application clients.	267
Removing application client resources	268
Running application clients.	268
launchClient tool	270
Application client troubleshooting tips	273

Chapter 7. Web Services Invocation Framework (WSIF): Enabling Web services.	279
Goals of WSIF	279
WSIF - Web services are more than just SOAP services	280
WSIF - Tying client code to a particular protocol implementation is restricting	280
WSIF - Incorporating new bindings into client code is hard	280
WSIF - Multiple bindings can be used in flexible ways	280
WSIF - Enabling a freer Web services environment promotes intermediaries	281
An overview of WSIF	281
WSIF architecture	282
Using WSIF with Web services that offer multiple bindings	282
WSIF and WSDL	282
WSIF usage scenarios.	283
Dynamic invocation	284
Using WSIF to invoke Web services	284
Using the WSIF providers	285
Developing a WSIF service.	296
Using complex types	305
Using the Java Naming and Directory Interface (JNDI).	306
Interacting with the J2EE container in WebSphere Application Server.	308
Running WSIF as a client	308
WSIF system management and administration	308
Maintaining the WSIF properties file	309
Enabling security for WSIF	309
WSIF troubleshooting tips	310
WSIF API	314
WSIF API reference: Creating a message for sending to a port	315
WSIF API reference: Finding a port factory or service.	316
WSIF API reference: Using ports	317
WSIF: Resources for learning	322

Chapter 8. IBM WebSphere UDDI Registry.	323
UDDI Registry terminology	323
UDDI Registry definitions	324
An overview of IBM UDDI Registries	324
Installing and setting up a UDDI Registry.	326
Installing the UDDI Registry into a deployment manager cell.	328
Installing the UDDI Registry into a single appserver	337
Reinstalling the UDDI Registry application	344
Removing the UDDI Registry application from a deployment manager cell	345
Removing the UDDI Registry application from a single appserver	345
Configuring the UDDI Registry	346
Configuring global UDDI properties.	346
Modifying the database userid and password	348

Configuring security properties	349
Configuring the UDDI User Console (GUI) for multiple language encoding support.	349
Customizing the UDDI User Console (GUI)	349
Configuring SOAP interface properties	349
Configuring SOAP properties with the AAT	350
Configuring SOAP properties in an already-deployed application	350
Configuring WebSphere to use HTTPS and SSL	351
Administering the UDDI Registry	351
Running the UDDI Registry	351
Backing up and restoring the UDDI Registry database	351
UDDI user console	352
Displaying the user console	356
SOAP Application Programming Interface for the UDDI Registry	356
Programming the UDDI SOAP API	357
SOAP API error handling tips in the UDDI Registry	357
UDDI Registry Application Programming Interface Inquiry API for the UDDI Registry	358
Publish API for the UDDI Registry	361
UDDI EJB Interface for the UDDI Registry	362
Datatypes package in the UDDI Registry	368
EJB interface methods in the UDDI Registry	371
UDDI troubleshooting tips	372
Turning on UDDI trace	375
Messages	375
UDAI (Web Services UDDI) messages	376
UDCF (Web Services UDDI) messages	376
UDDA (Web Services UDDI) messages	377
UDDM (Web Services UDDI) messages	377
UDEJ (Web Services UDDI) messages	377
UDEX (Web Services UDDI) messages	377
UDIN (Web Services UDDI) messages	377
UDLC (Web Services UDDI) messages	400
UDPR (Web Services UDDI) messages	400
UDRS (Web Services UDDI) messages	401
UDSC (Web Services UDDI) messages	401
UDSP (Web Services UDDI) messages	401
UDUC (Web Services UDDI) messages	402
UDUT UDDI Utility Tools messages	404
UDUU (Web Services UDDI) messages	413
Running the UDDI Samples	413
Installation Verification Program (IVP)	413
Reporting problems with the IBM WebSphere UDDI Registry	414
Feedback	415

Chapter 9. Class loading 417

Class loaders	418
Class loader collection	421
Classloader ID	422
Classloader Mode	422
Class loader settings	422
Migrating the class-loader Module Visibility Mode setting.	422
Class loading: Resources for learning	423

Chapter 10. Using EJB query 425

EJB query language	425
Example: EJB queries	426
FROM clause	428
Inheritance in EJB query.	429
Path expressions	429
WHERE clause	430
Scalar functions	438
Aggregation functions	441
SELECT clause	442
ORDER BY clause	443
Subqueries	444
EJB query restrictions.	444
EJB Query: Reserved words	445
EJB query: BNF syntax	445
Comparison of EJB 2.0 specification and WebSphere query language.	447

Chapter 11. Internationalizing applications 449

Internationalization	449
Identifying localizable text	450
Creating message catalogs	450
Composing language-specific strings	451
Localization API support	451
LocalizableTextFormatter class.	452
Creating a formatter instance	455
Setting optional localization values	456
Generating localized text	458
Preparing the localizable-text package for deployment	458
LocalizableTextEJBDeploy command	459
Internationalization: Resources for learning	460

Chapter 12. Using the transaction service 461

Transaction support in WebSphere Application Server	461
Resource manager local transaction (RMLT)	463
Global transactions	463
Local transaction containment (LTC).	464
Local and global transaction considerations	464
Developing components to use transactions	465
Setting transactional attributes in the deployment descriptor	466
Using bean-managed transactions	467
Classifying WebSphere transaction workload for WLM	468
Configuring transaction properties for an application server	470
Transaction service settings.	471
Using local transactions	474
Managing active transactions	476
Interoperating transactionally between application servers	477
Troubleshooting transactions	478
Transaction service exceptions.	479
UserTransaction interface - methods available	480

Chapter 13. Using naming 481

Naming	482
------------------	-----

New features for name space support	482
Name space logical view	483
Initial context support	486
Lookup names support in deployment descriptors and thin clients.	487
JNDI support in WebSphere Application Server	490
Developing applications that use JNDI	490
Example: Getting the default initial context	492
Example: Getting an initial context by setting the provider URL property	496
Example: Setting the provider URL property to select a different root context as the initial context	498
Example: Looking up an EJB home with JNDI	499
Example: Looking up a JavaMail session with JNDI	501
JNDI interoperability considerations.	501
JNDI caching	503
JNDI cache settings	504
Example: Controlling JNDI cache behavior from a program	506
JNDI name syntax.	506
INS name syntax	507
JNDI to CORBA name mapping considerations	507
Example: Setting the syntax used to parse name strings.	507
Developing applications that use CosNaming (CORBA Naming interface).	508
Example: Getting an initial context with CosNaming	508
Example: Looking up an EJB home with CosNaming	511
Configured name bindings	513
Name space federation	515
Name space bindings.	516
Configuring and viewing name space bindings	517
String binding settings	517
CORBA object binding settings	518
Indirect lookup binding settings	518
EJB binding settings	519
Name space binding collection	520
Configuring name servers	520
Name server settings	521
Troubleshooting name space problems	521
dumpNameSpace tool	521
Example: Invoking the name space dump utility	524
Name space dump utility for java:, local: and server name spaces	525
Example: Invoking the name space dump utility for java: and local: name spaces	527
Name space dump sample output	528
Naming and directories: Resources for learning	530
Chapter 14. Managing user profiles	531
User profile	531

UserProfileManager class	532
User profile development options	532
Example: UPServletExample.java	533
Example: UserProfileExtendedSample.java.	535
Example: UPServletExampleExtended.java.	536
Example: UserProfileExtended.java	538
Example: UPServletExtended.java	539
userprofile.xml	541

Chapter 15. Assembling applications 543

Application assembly and J2EE applications	544
Archive support in Version 5.0	545
Starting the Application Assembly Tool (AAT)	545
Migrating application modules from J2EE 1.2 to J2EE 1.3	546
earconvert tool	547
Assembling new or modifying existing modules	548
Adding files to assembled modules	550
Resource environment reference assembly settings	551
Resource Adapter Archive file assembly settings	552
Saving applications after assembly	555
Verifying archive files	556
Application assembly performance checklist	557
Generating code for deployment	557
ejbdeploy tool	558
ejbdeploy syntax — relationship to Application Assembly Tool options	558
Application Assembly Tool: Resources for learning	559

Chapter 16. Deploying and managing applications 561

Enterprise applications	561
Installing a new application	561
Preparing for application installation settings	566
Example: Installing an EAR file using the default bindings	569
Enterprise application collection	570
Name	570
Status	570
Enterprise application settings.	570
Starting and stopping applications	573
Exporting applications	574
Exporting DDL files	574
Updating applications	574
Hot deployment and dynamic reloading	575
Uninstalling applications	583
Deploying and managing applications: Resources for learning	584

Notices 585

Trademarks and service marks 587

How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information.

- To send comments on articles in the WebSphere Application Server Information Center
 1. Display the article in your Web browser and scroll to the end of the article.
 2. Click on the **Feedback** link at the bottom of the article, and a separate window containing an e-mail form appears.
 3. Fill out the e-mail form as instructed, and click on **Submit feedback** .
- To send comments on PDF books, you can e-mail your comments to: **wasdoc@us.ibm.com** or fax them to 919-254-0206.

Be sure to include the document name and number, the WebSphere Application Server version you are using, and, if applicable, the specific page, table, or figure number on which you are commenting.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Chapter 1. Welcome to Applications

The following items comprise the application programming model, including numerous services available to support deployed applications.

Web modules

Use Web components such as servlets and JavaServer Pages files to develop dynamic Web sites. Product extensions to the open source servlet and JSP APIs enhance standard features, and provide additional functionality.

Web modules consist of the following application components, each performing a different function:

- HTML and JSP pages provide the user interface and program logic
- Servlets coordinate work between other components of the application
- Bean Scripting Framework enables users to implement JavaScript code in JSP applications

HTTP sessions are a key area of product support for Web modules. By **managing HTTP sessions** for your Web applications, you can personalize a Web site for individual customers. A session is a series of requests to a servlet, originating from the same user at the same browser. Managing HTTP sessions allows servlets running in a Web container to keep track of individual users. For example, a servlet might use sessions to provide "shopping carts" to on-line shoppers. Suppose the servlet is designed to record the items each shopper indicates he or she will purchase from the Web site. It is important that the servlet be able to associate incoming requests with particular shoppers. Otherwise, the servlet might mistakenly add choices of Shopper 1 to the cart of Shopper 2.

EJB modules

IBM WebSphere Application Server provides broad support for enterprise beans, including the Enterprise JavaBeans (EJB) 2.0 specification. The EJB 2.0 specification introduces a container-managed persistence (CMP) 2.0 component model, which provides a number of improvements to aid developer productivity and application performance. In addition, this product continues to fully support enterprise beans written to the CMP 1.1 programming model and deployed in previous versions of this product; applications can use CMP 1.1 beans, CMP 2.0 beans, or a mixture of both. CMP 1.1 beans can be directly carried forward in an EJB 1.1 `ejb-jar` module or may be repackaged and combined with CMP 2.0 beans in an EJB 2.0 module.

For EJB 2.0 modules, a feature introduced in Version 5 of this product, called **access intent** policies, eases the management of interactions between CMP beans and their underlying datastores. Each policy sets such data access characteristics such as access type (read or update) and transaction isolation that affect the locking of resources, letting you choose the level of data integrity and performance for your application.

Several excellent trade books that cover EJB 2.0 and the CMP 2.0 persistence model are already available. A good way to locate some of these is to visit your favorite online bookstore and search on the term *Enterprise JavaBeans*. For a more basic orientation, see *Enterprise beans: Resources for learning*.

Your application development might include **asynchronous messaging**, which the product supports as a method of communication based on the Java Message Service (JMS) programming interface.

The base JMS support enables IBM WebSphere Application Server applications to exchange messages asynchronously with other JMS clients by using JMS destinations (queues or topics). An application can explicitly poll for messages on a destination.

The product also provides a message listener service that applications can use to automatically retrieve messages from JMS destinations for processing by message-driven beans, without the application having to explicitly poll JMS destinations.

Refer to:

- Asynchronous messaging with WebSphere - an overviewAn overview of WebSphere asynchronous messaging.
- Using JMS and messaging in applicationsImplementing WebSphere J2EE applications that use JMS.

Client modules

The Imaginary Buffer Line product provides a CD-ROM and installation program for installing application clients without installing the entire application server. This smaller footprint is useful if you want to run client applications on multiple client machines. For more information, see *Installing application clients*.

Application clients follow several programming models with unique requirements and suitability for different types of applications. Most of these models are only available when you install the product. Models include:

ActiveX application client

Supported Windows platforms only; client only

Applet client

Supported Windows platforms only; client only

J2EE application client

All supported server platforms; all supported client platforms

Pluggable application client

Supported Windows platforms only, Sun JRE 1.3.1_03 or later (but not version 1.4); thin application client programming model only (no J2EE programming model); client only

Thin application client

All supported client platforms

Web services

The Web services components included with this product version build upon the Apache Simple Object Access Protocol (SOAP) 2.3-based capabilities delivered with Version 4.0.x of the product.

Version 5.0 introduced an open source implementation for a Web Services Invocation Framework (WSIF).

An additional Web services component, "IBM WebSphere Web Services for J2EE Technology Preview" is available for use with Version 5. It is available as a separate download from:
<http://www7b.boulder.ibm.com/wsdd/downloads/techpreviews.html>. This

component is shipped with the WebSphere Application Server for z/OS product. Use the WebSphere Application Server for z/OS ISPF customization dialog to enable this component.

The Web Services Technology Preview supports emerging Java Web services standards like JAX-RPC and Web services for J2EE. It is recommended that new development efforts use the Web Services Technology Preview and follow these standards.

Additional features, such as UDDI Registry, and Web Services Gateway are described in Welcome to Servers.

Application services

IBM WebSphere Application Server provides essential services to ease the building of dynamic and flexible e-business applications. These services support and extend the open standards of J2EE and Web services, with a focus on application reuse and integration.

- **Class loading**

The WebSphere Application Server product provides several class-loading modes, policies, and features to enable you to deploy and run your applications successfully. An application server provides an Application class-loader policy that enables you to control the isolation of applications in a server. If you want applications to share classes, choose the SINGLE policy; otherwise choose the MULTIPLE policy, which isolates the class loaders for each application.

Similarly, at the application level, you can choose a WAR class-loader policy that configures the isolation of Web modules within an application. If you choose the policy APPLICATION, then each Web module in your application can see the classes of other Web modules. A policy of MODULE creates a separate class loader for each Web module, resulting in isolation for each the classes of each Web module.

The class-loader mode setting, which you can configure at the server, application, or Web module level depending on your class-loader policy, enables you to control whether application class loaders override classes contained in base run-time class loaders. By default, the WebSphere Application Server class loaders have a class-loader mode of PARENT_FIRST, which is the standard JDK mode and does not allow the application class loader to override classes. You must take care when using the PARENT_LAST class-loader mode to make all dependent classes available within the application or you might get LinkageErrors or other class-loader exceptions. For example, if you provide a newer version of the Xerces.jar file and your application is using XSLT, you must also provide a xalan.jar file within your application.

- **Shared library**

Version 5.0 of WebSphere Application Server introduces the concept of a shared library. A shared library is a CLASSPATH and a symbolic name for that classpath. You define shared libraries at the cell, node, or server level and then associate the shared libraries either with an application server (making the classes available to all applications in the server) or with individual applications (making the classes available only to the referencing application). This mechanism provides a convenient way to make libraries of classes available to your applications outside of a standard J2EE enterprise application (EAR) file for easier version management and space efficiency.

- **Internationalization support**

If your application component must support multiple locales, the **localizable-text** API can help both developers and administrators through

central management of displayed strings. The developer separates strings into a message catalog, which is then translated into the other languages required. All message catalogs are then deployed with the application component. From then on, the administrator can add or update message catalogs centrally as required. See Internationalizing applications.

- **Transactions**

IBM WebSphere Application Server applications can use transactions to coordinate multiple updates to resources as atomic units (as indivisible units of work) such that all or none of the updates are made permanent. The way that applications use transactions depends on the type of application component, as follows:

- 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 (servlets) use bean-managed transactions

The product is a transaction manager that supports the coordination of resource managers through their XAResource interface and participates in distributed global transactions with other OTS 1.2 compliant transaction managers (for example J2EE 1.3 application servers). Applications can also be configured to interact with databases, JMS queues, and JCA connectors through their local transaction support when distributed transaction coordination is not required.

Resource managers that offer transaction support can be categorized into those that support 2-phase coordination (by offering an XAResource interface) and those that support only 1-phase coordination (for example through a LocalTransaction interface). The IBM WebSphere Application Server transaction support provides coordination, within a transaction, for any number of 2-phase capable resource managers. It also enables a single 1-phase capable resource manager to be used within a transaction in the absence of any other resource managers, although a WebSphere transaction is not necessary in this case. With the Last Participant Support of Enterprise Extensions, you can coordinate the use of a single 1-phase commit (1PC) capable resource with any number of 2-phase commit (2PC) capable resources in the same global transaction. At transaction commit, the 2-phase commit resources are prepared first using the 2-phase commit protocol, and if this is successful the 1-phase commit-resource is then called to commit (1-phase). The 2-phase commit resources are then committed or rolled back depending on the response of the 1-phase commit resource.

The ActivitySession service of Enterprise Extensions provides an alternative unit-of-work (UOW) scope to that provided by global transaction contexts. It is a distributed context that can be used to coordinate multiple 1-phase resource managers. The product EJB container and deployment tooling support ActivitySessions as an extension to the J2EE programming model. Enterprise beans can be deployed with lifecycles that are influenced by ActivitySession context, as an alternative to transaction context. An application can then interact with a resource manager through its LocalTransaction interface for the period of a client-scoped ActivitySession rather than just the duration of an EJB method.

- **Naming**

Naming clients use **Naming Services** primarily to access objects, such as EJB homes, associated with applications installed on IBM WebSphere Application Server. Objects are made available to clients by being bound into a name space. A name space is under the control of a name server. In this product, there are potentially many name servers, and the name spaces controlled by the various

name servers are federated together to form the view of a single name space. Each name server presents the same logical view of the federated name spaces. Name servers provided by this product are a CORBA CosNaming implementation. IBM WebSphere Application Server provides a CosNaming JNDI plug-in which enables clients to access the name servers through the JNDI interface. Clients to EJB applications typically use JNDI to perform Naming operations. Clients may access the name servers directly through the CORBA programming model. The CosNaming interface is part of the CORBA programming model. CORBA clients which need to access EJB homes or some other objects bound to the name space would typically use the CORBA CosNaming interface to perform Naming operations.

- **Dynamic cache**

Dynamic cache improves application performance by caching outputs and contents of outputs of Servlets, JavaServer Pages (JSP) files, Web Services, and Commands. On subsequent client requests to the same applications, Dynamic cache intercepts these calls and responds by serving the output or the contents of output from the cache.

Dynamic cache in this product version includes:

- **Servlet/JSP files caching**

- This caches output of dynamic servlets and JSP files by working with Java virtual machine of the application server by intercepting calls to service methods and serving Web pages from the cache. This improves server response time, throughput and scalability.

- **Command caching**

- Commands that are written to the Command Architecture encapsulate business logic tasks and provide a standard way to invoke the business logic request. Command objects need to implement CacheableCommand interface instead of TargetableCommand interface to cache. Like in servlets and JSP caching, requests to execute business logic in the command is intercepted by the cache. If a command with the same request attributes are available in cache, output properties are copied from the cached instance to the requested instance and returned without executing the business logic again.

- **Web Services caching**

- Web service responses can be cached just like servlet and JSP results. These requests are intercepted and cache ID computed based on how the cache ID rules are specified in the cache policy. Hash of the whole SOAPEnvelope can be used as a cache ID or it can be parsed and service, operation and parameters to these operations used as cache ID. If a cache entry is not found for the computed cache id, the request is forwarded to the SOAP engine and the result is cached.

- **Edge Side Include caching**

- This provides the ability to cache, assemble and deliver dynamic web pages at the edge of the enterprise network. Edge Side Includes (ESI) is a simple markup language which enables dynamic web pages (which by themselves are not so cache efficient) to be broken down into cacheable fragments. These fragments are then cached on the edge of the network and assembled into a single page upon user requests.

- **Distributed caching**

- Cache contents can be shared and replicated among servers by Dynamic Caching using an underlying JMS based message broker system, Data Replication Service (DRS). Sharing characteristics of individual cache entry is configured using the cache policy specification.

- **User profiles**

Managing **user profiles** allows a company to maintain database tables containing fields for demographic data of individual customers or other users on the company system. For example, when a user repeatedly logs onto a Web site that supports user profiles, the Web site can display headlines and advertising tailored to the shopping preferences of that user. The site can address the user by his or her logon name. User profile API is deprecated in the current release.

Assembly tools

The Application Assembly Tool (AAT) and a command line deployment tool are provided for packaging your application code components into the needed modules for deployment onto the server.

See Assembling applications with the AAT.

EAR files are composed of the following archives:

- Enterprise bean JAR files (known as EJB modules)
- Web application WAR files (known as Web modules)
- Application client JAR files (known as Application clients)
- Resource adapter RAR files (known as resource adapter modules)
- Optionally, additional JAR files containing dependent classes or other components required by the application

The standard file extension of an Enterprise application file is .ear.

For a discussion of archives and Web components supported in V5 by the Application Assembly Tool, see Archive support in Version 5.0.

See Assembling or packaging for more information.

Deployment

The WebSphere Application Server administrative console is one of the tools you can use to install your modules onto the Application Server, and then manage the installed modules in their respective containers.

Chapter 2. Using Web applications

A developer creates the files comprising a Web application, and then assembles the Web application components into a Web module. Next, the deployer (typically the developer in a unit-testing environment or the administrator in a production environment) installs the Web application on the server.

1. **(Optional)** Migrate existing Web applications to run in the new version of WebSphere.
2. Design the Web application and develop its code artifacts: Servlets, JavaServer Pages (JSP) files, and static files, as for example, images and Hyper Text Markup Language (HTML) files. See the "Resources for learning" article for links to design documentation.
3. **(Optional)** Implement JavaScript within JSP tags using the Bean Scripting Framework (BSF).
4. Develop the Web application, using WebSphere Application Server extensions to enhance its functionality.
5. Assemble the Web application into a Web module.
Web module assembly properties might include the ability to:
 - Configure servlet page lists
 - Configure servlet filters
 - Serve servlets by class name
 - Enable file serving
6. Deploy the Web module or application module that contains the Web application.
Following deployment, you might find it handy to use the tool that enables batch compiling of the JSP files for quicker initial response times.
7. **(Optional)** Troubleshoot your Web application.
8. **(Optional)** Modify the default Web container configuration in the application server in which you deployed the Web module or application module containing the Web application.
9. **(Optional)** Manage the deployed Web application..

Web applications

A Web application is comprised of one or more related servlets, JavaServer Pages technology (JSP files), and Hyper Text Markup Language (HTML) files that you can manage as a unit.

The files in a Web application are related in that they work together to perform a business logic function.

For example, one of the WebSphere Application Server samples is a *Simple Greeting* Web application. This application, comprised of a servlet and Web pages, greets new users when the application is accessed.

The Web application is a concept supported by the Java Servlet Specification. Web applications are typically packaged as .war files.

web.xml file

The web.xml file provides configuration and deployment information for the Web components that comprise a Web application. Examples of Web components are servlet parameters, servlet and JavaServer Pages (JSP) definitions, and Uniform Resource Locators (URL) mappings.

The servlet 2.3 specification dictates the format of the web.xml file, which makes this file portable among Java Two Enterprise Edition (J2EE) compliant products.

Location

The web.xml file must reside in the WEB-INF directory under the context of the hierarchy of directories that exist for a Web application. For example, if the application is `client.war`, then the web.xml file is placed in the `install_root/client.war/WEB-INF` directory.

Usage notes

- Is this file read-only?

No

- Is this file updated by a product component?

This file is updated by the Application Assembly Tool (AAT).

- If so, what triggers its update?

The AAT updates the web.xml file when you assemble Web components into a Web module, or when you modify the properties of the Web components or the Web module.

- How and when are the contents of this file used?

WebSphere Application Server functions use information in this file during the configuration and deployment phases of Web application development.

Sample file entry

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">
<web-app id="WebApp_1">
  <display-name>Persistence Manager Web Client</display-name>
  <description>Peristence Manager Web Client</description>
  <servlet id="Servlet_1">
    <servlet-name>CustomerLocalServlet</servlet-name>
    <description>Local Customer Servlet</description>
    <servlet-class>CustomerLocalServlet</servlet-class>
  </servlet>
  <servlet id="Servlet_2">
    <servlet-name>CustomerServlet</servlet-name>
    <description>Remote Customer Servlet</description>
    <servlet-class>CustomerServlet</servlet-class>
  </servlet>
  <servlet id="Servlet_3">
    <servlet-name>CreditCardServlet</servlet-name>
    <description>Credit Card Servlet - PM Verification</description>
    <servlet-class>CreditCardServlet</servlet-class>
  </servlet>
  <servlet-mapping id="ServletMapping_1">
    <servlet-name>CustomerLocalServlet</servlet-name>
    <url-pattern>/CustomerLocal</url-pattern>
  </servlet-mapping>
  <servlet-mapping id="ServletMapping_2">
    <servlet-name>CustomerServlet</servlet-name>
    <url-pattern>/Customer</url-pattern>
  </servlet-mapping>
</web-app>
```

```

</servlet-mapping>
<servlet-mapping id="ServletMapping_3">
  <servlet-name>CreditCardServlet</servlet-name>
  <url-pattern>/CreditCard</url-pattern>
</servlet-mapping>
<welcome-file-list id="WelcomeFileList_1">
  <welcome-file>index.html</welcome-file>
</welcome-file-list>
<security-role id="SecurityRole_1">
  <description>Everyone role</description>
  <role-name>Everyone Role</role-name>
</security-role>
<security-role id="SecurityRole_2">
  <description>AllAuthenticated role</description>
  <role-name>All Role</role-name>
</security-role>
<security-role id="SecurityRole_3">
  <description>Deny all access role</description>
  <role-name>DenyAllRole</role-name>
</security-role>
</web-app>

```

Migrating Web application components

Supported open specification levels in WebSphere Application Server Version 5 are documented in article, [Migrating](#).

Migration of Web applications deployed in WebSphere Application Server Version 4.x is not necessary; version 2.2 of the servlet specification and version 1.1 of the JavaServerPages (JSP) specification are still supported. However, where there are behavioral differences between the Java Two Enterprise Edition (J2EE) 1.2 and J2EE 1.3 specifications, bear in mind that J2EE 1.3 specifications are implemented in WebSphere Application Server Version 5 and will override any J2EE 1.2 behaviors.

Migration of Web applications deployed in WebSphere Application Server Version 4.0.1 is not necessary; version 2.2 of the servlet specification and version 1.1 of the JavaServerPages (JSP) specification are still supported. However, where there are behavioral differences between the Java Two Enterprise Edition (J2EE) 1.2 and J2EE 1.3 specifications, bear in mind that J2EE 1.3 specifications are implemented in WebSphere Application Server Version 5 and will override any J2EE 1.2 behaviors.

Servlet migration might be a concern if your application:

- implements a WebSphere internal servlet to bypass a WebSphere Application Server Version 4.x single application path restriction.
- implements a WebSphere internal servlet to bypass a WebSphere Application Server Version 4.0.1 single application path restriction.
- extends a PageListServlet that relies on configuration information in the servlet configuration XML file.
- uses a servlet to generate Hyper Text Markup Language (HTML) output.
- calls the `response.sendRedirect()` method for a servlet using the `encodeRedirectURL` function or executing within a non-context root.

JSP migration might be a concern if your application references JSP page implementation classes in unnamed packages, or if you install WebSphere Application Server Version 4.x EAR files (deployed in Version 4.x with the JSP Precompile option), in Version 5.

JSP migration might be a concern if your application references JSP page implementation classes in unnamed packages, or if you install WebSphere Application Server Version 4.0.1 EAR files (deployed in Version 4.0.1 with the JSP Precompile option), in Version 5.

Follow these steps if migration issues apply to your Web application:

1. Use WebSphere Application Server Version 5 package names for any WebSphere Application Server Version 4.x internal servlets, which are implemented in your application.

In WebSphere Application Server Version 4.x, Web modules with a context root setting of / are not supported. Accessing Web modules with this root context results in HTTP 404 - File not Found errors.

To bypass the errors, and to enable the serving of static files from the root context, WebSphere Application Server Version 4.x users are advised to add the servlet class, `com.ibm.servlet.engine.webapp.SimpleFileServlet`, to their Web module.

The Version 4.x single path limitation does not exist in Version 5. However, users who choose to use the `com.ibm.servlet.engine.webapp.SimpleFileServlet` in Version 5 must do one of the following:

- Rename `com.ibm.servlet.engine.webapp.SimpleFileServlet` to `com.ibm.ws.webcontainer.servlet.SimpleFileServlet`.
- Open the EAR file in the Application Assembly Tool (AAT) and enable the `SimpleFileServlet` static file setting.

The following list identifies the other internal servlets affected by the Version 5 package name change:

- `DefaultErrorReporter`
- `AutoInvoker`

Use the Version 5 package name, `com.ibm.ws.webcontainer.servlet.<servlet class name>` for these servlets.

2. Use WebSphere Application Server Version 5 package names for any WebSphere Application Server Version 4.0.1 internal servlets, which are implemented in your application.

In WebSphere Application Server Version 4.0.1, Web modules with a context root setting of / are not supported. Accessing Web modules with this root context results in HTTP 404 - File not Found errors.

To bypass the errors, and to enable the serving of static files from the root context, WebSphere Application Server Version 4.0.1 users are advised to add the servlet class, `com.ibm.servlet.engine.webapp.SimpleFileServlet`, to their Web module.

The Version 4.0.1 single path limitation does not exist in Version 5. However, users who choose to use the `com.ibm.servlet.engine.webapp.SimpleFileServlet` in Version 5 must do one of the following:

- Rename `com.ibm.servlet.engine.webapp.SimpleFileServlet` to `com.ibm.ws.webcontainer.servlet.SimpleFileServlet`.
- Open the EAR file in the Application Assembly Tool (AAT) and enable the `SimpleFileServlet` static file setting.

The following list identifies the other internal servlets affected by the Version 5 package name change:

- `DefaultErrorReporter`

- AutoInvoker

Use the Version 5 package name, `com.ibm.ws.webcontainer.servlet.<servlet class name>` for these servlets.

3. Use the WASPostUpgrade tool to migrate servlets that extend `PageListServlet` and rely on configuration information in the associated XML servlet configuration file. In Version 4.x, the XML servlet configuration file provides configuration data for page lists and augments servlet configuration information. This file is named as either `servlet_class_name.servlet` or `servlet_name.servlet`, and is stored in the same directory as the servlet class file.

The XML servlet configuration file is not supported in WebSphere Application Server Version 5.

4. Migrate servlets that extend `PageListServlet` and rely on configuration information in the associated XML servlet configuration file. In Version 4.0.1, the XML servlet configuration file provides configuration data for page lists and augments servlet configuration information. This file is named as either `servlet_class_name.servlet` or `servlet_name.servlet`, and is stored in the same directory as the servlet class file.

The XML servlet configuration file is not supported in WebSphere Application Server Version 5. The direct use of the servlet has been deprecated. The `PageList` servlet function is still available but is configured as part of the servlet extension configuration in the WAR file.

5. Set a content type if your servlet generates Hyper Text Markup Language (HTML) output.

The default behavior of the Web container changed in WebSphere Application Server Version 5. If the servlet developer does not specify a content type in the servlet then the container is forbidden to set one automatically. Without an explicit content type setting, the content type is set to null. The Netscape browser displays HTML source as plain text with a null content type setting.

To resolve this problem, do one of the following:

- Explicitly set a content type in your servlet.
- Open the WAR file in the Application Assembly Tool (AAT) and enable the `autoResponseEncoding` static file setting.

6. Set the Java environment variable, `com.ibm.websphere.sendredirect.compatibility`, to **true** if you want your URLs interpreted relative to the application root.

The default value of the Java environment variable `com.ibm.websphere.sendredirect.compatibility` changed in WebSphere Application Server Version 5. In Version 4, the default setting of this variable is true. In Version 5, the setting is false.

When this variable is set to false, if a URL has a leading slash, the URL is interpreted relative to the Web module/application root. However, if the URL does not have a leading slash, it is interpreted relative to the Web container root (also known as the Web server document root). Therefore, if an application has a WAR file that has a context root of `myPledge_app` and a servlet that has a servlet mapping of `/Intranet/`, a JSP file in the WAR file cannot access the servlet when its `encodeRedirectURL` is set to `/Intranet/myPledge`. The JSP file can access the servlet if the `encodeRedirectURL` is set to `myPledge_app/Intranet/myPledge`, or if the

`com.ibm.websphere.sendredirect.compatibility` variable is set to true.

See the [Setting the sendredirect variable](#) article for more information.

7. Use the WASPostUpgrade tool to migrate WebSphere Version 4.x enterprise applications to Version 5.

Note: The WebSphere Application Server Version 4.x JSP page implementation class files are not compatible with the WebSphere Application Server Version 5 JSP container.

The WASPostUpgrade tool automatically precompiles JSP files, which ensures the JSP page implementation class files are compatible with Version 5.

If you install Version 4.x EAR files, deployed with the JSP Precompile option, in Version 5, and you choose not to follow the migration path, do one of the following:

- Select the Pre-compile JSP option in the administrative console Install New Application window.

See article Installing a new application for more information.

- Specify the `-preCompileJSPs` option when using the Wsadmin tool.

8. Migrate WebSphere Version 4.0.1 enterprise applications to Version 5.

Note: The WebSphere Application Server Version 4.0.1 JSP page implementation class files are not compatible with the WebSphere Application Server Version 5 JSP container.

You must do one of the following:

- Select the Pre-compile JSP option in the administrative console Install New Application window.

See article Installing a new application for more information.

- Specify the `-preCompileJSPs` option when using the Wsadmin tool.

9. Import your classes if your application uses unnamed packages.

Section 8.2 of the JSP 1.2 specification states:

The JSP container creates a JSP page implementation class for each JSP page. The name of the JSP page implementation class is implementation dependent. The JSP page implementation object belongs to an implementation-dependent named package. The package used may vary between one JSP and another, so minimal assumptions should be made. The unnamed package should not be used without an explicit *import* of the class.

For example, if `myBeanClass` is in the unnamed package, and you reference it in a `jsp:useBean` tag, then you must explicitly import `myBeanClass` with the page directive `import` attribute, as shown in the following example:

```
<%@page import="myBeanClass" %>
.
.
.
<jsp:useBean id="myBean" class="myBeanClass" scope="session"/>
```

In WebSphere Application Server Version 5, the JSP engine creates JSP page implementation classes in the `org.apache.jsp` package. If a class in the unnamed package is not explicitly imported, then the javac compiler assumes the class is in package `org.apache.jsp`, and the compilation fails.

Note: Avoid using the unnamed package altogether because of a change made in JDK 1.4 that will affect the JSP 2.0 specification. WebSphere Application Server Version 5 ships with JDK 1.3.1, so this is not an issue with the Version 5 JSP engine, but it will become an issue in future releases.

The *Incompatibilities* section of the version 1.4.Java 2 Platform, Standard Edition (J2SE) documentation states:

The compiler now rejects import statements that import a type from the unnamed namespace. Previous versions of the compiler would accept such import declarations, even though they were arguably not allowed by the language (because the type name appearing in the import clause is not in scope). The specification is being clarified to state clearly that you cannot have a simple

name in an import statement, nor can you import from the unnamed namespace.

To summarize, the syntax:

```
import SimpleName;
```

is no longer legal. Nor is the syntax

```
import ClassInUnnamedNamespace.Nested;
```

which would import a nested class from the unnamed namespace.

To fix such problems in your code, move all of the classes from the unnamed namespace into a named namespace.

See "Resources for learning" for links to the J2SE, JSP, and Servlet specification documentation.

Default Application

The IBM WebSphere Application Server provides a default configuration that allows administrators to easily verify that the Application Server is running. When the product is installed, it includes an application server called *server1* and an enterprise application called *Default Application*.

Default Application contains a Web Module called *DefaultWebApplication* and an enterprise bean JAR file called *Increment*. The *Default Application* provides a number of servlets, described below. These servlets are available in the product.

For additional code examples, visit the Samples Gallery. Learn how to locate and install the Samples Gallery by viewing the Samples Gallery reference page.

The URL for accessing Samples is: <http://localhost:9080/WSamples/>

Snoop

Use the Snoop servlet to retrieve information about a servlet request. This servlet returns the following information:

- Servlet initialization parameters
- Servlet context initialization parameters
- URL invocation request parameters
- Preferred client locale
- Context path
- User principal
- Request headers and their values
- Request parameter names and their values
- HTTPS protocol information
- Servlet request attributes and their values
- HTTP session information
- Session attributes and their values

The Snoop servlet includes security configuration so that when WebSphere Security is enabled, clients must supply a user ID and password to execute the servlet.

The URL for the Snoop servlet is: <http://localhost:9080/snoop/>.

HelloHTML

Use the HelloHTML pervasive servlet to exercise the PageList support provided by the WebSphere Web container. This servlet extends the PageListServlet, which

provides APIs that allow servlets to call other Web resources by name or, when using the *Client Type detection* support, by type.

You can invoke the Hello servlet from an HTML browser, speech client, or most Wireless Application Protocol (WAP) enabled browsers using the URL:
`http://localhost:9080/HelloHTML.jsp`.

HitCount

Use the HitCount Demonstration application to demonstrate incrementing a counter using a variety of methods, including:

- A servlet instance variable
- An HTTP session
- An enterprise bean

You can instruct the servlet to execute any of these methods within a transaction that you can omit or roll back. If the transaction is committed, the counter is incremented. If the transaction is rolled back, the counter is not incremented.

The enterprise bean method uses a Container- Managed Persistence enterprise bean that persists the counter value to a Cloudscape database. This enterprise bean is configured to use the Default Datasource, which is set to the DefaultDB database.

When using the enterprise bean method, you can instruct the servlet to look up the enterprise bean, either in the WebSphere global namespace, or in the namespace local to the application.

The URL for the HitCount application is: `http://localhost:9080/HitCount.jsp`.

Servlets

Servlets are Java programs that use the Java Servlet Application Programming Interface (API). You must package servlets in a Web ARchive (WAR) file or Web module for deployment to the application server.

Servlets run on a Java-enabled Web server and extend the capabilities of a Web server, similar to the way applets run on a browser and extend the capabilities of a browser.

Servlets can support dynamic Web page content, provide database access, serve multiple clients at one time, and filter data.

For the purposes of IBM WebSphere Application Server, discussions of servlets focus on Hyper Text Transfer Protocol (HTTP) servlets, which serve Web-based clients.

Developing servlets with WebSphere Application Server extensions

Several WebSphere Application Server extensions are provided for enhancing your servlets. This task provides a summary of the extensions that you can utilize.

1. Review the supported specifications.

Create Java components, referring to the Servlet specifications from Sun Microsystems.

See Resources for learning for links to coding specifications and examples.

The application server includes its own packages that extend and add to the Java Servlet Application Programming Interface (API). These extensions and additions make it easier to manage session states, create personalized Web pages, generate better servlet error reports, and access databases. Locate the Javadoc for the application server APIs in the product `install_root\web\apidocs` directory.

All the public WebSphere Application Server APIs are located in the `com.ibm.websphere...` packages.

2. Use your favorite integrated development environment (IDE), or a text editor, to develop or migrate code artifacts that meet the specifications.
3. Test the code artifacts.

Assemble your code artifacts into a Web module as a prerequisite to deploying the code to the application server.

Application lifecycle listeners and events

Application lifecycle listeners and events, now part of the Servlet API, enable you to notify interested listeners when servlet contexts and sessions change. For example, you can notify users when attributes change and if sessions or servlet contexts are created or destroyed.

The lifecycle listeners give the application developer greater control over interactions with `ServletContext` and `HttpSession` objects. Servlet context listeners manage resources at an application level. Session listeners manage resources associated with a series of requests from a single client. Listeners are available for lifecycle events and for attribute modification events. The listener developer creates a class that implements the `javax` listener interface, corresponding to the desired listener functionality.

At application startup time, the container uses introspection to create an instance of your listener class and registers it with the appropriate event generator.

When a servlet context is created, the `contextInitialized` method of your listener class is invoked, which creates the database connection for the servlets in your application to use, if this context is for your application.

When the servlet context is destroyed, your `contextDestroyed` method is invoked, which releases the database connection, if this context is for your application.

Listener classes for servlet context and session changes

The following methods are defined as part of the `javax.servlet.ServletContextListener` interface:

- `void contextInitialized(ServletContextEvent)` - Notification that the Web application is ready to process requests.

Place code in this method to see if the created context is for your Web application and if it is, allocate a database connection and store the connection in the servlet context.

- `void contextDestroyed(ServletContextEvent)` - Notification that the servlet context is about to shut down.

Place code in this method to see if the created context is for your Web application and if it is, close the database connection stored in the servlet context.

Two new listener interfaces are defined as part of the `javax.servlet` package:

- `ServletContextListener`

- ServletContextAttributeListener

One new filter interface is defined as part of the javax.servlet package:

- FilterChain interface - methods: doFilter()

Two new event classes are defined as part of the javax.servlet package:

- ServletContextEvent
- ServletContextAttributeEvent

Three new listener interfaces are defined as part of the javax.servlet.http package:

- HttpSessionListener
- HttpSessionAttributeListener
- HttpSessionActivationListener

One new event class is defined as part of the javax.servlet.http package:

- HttpSessionEvent

Example: com.ibm.websphere.DBConnectionListener.java

The following example shows how to create a servlet context listener:

```
package com.ibm.websphere;

import java.io.*;
import javax.servlet.*;

public class DBConnectionListener implements ServletContextListener
{
    // implement the required context init method
    void contextInitialized(ServletContextEvent sce)
    {
    }

    // implement the required context destroy method
    void contextDestroyed(ServletContextEvent sce)
    {
    }
}
```

Servlet filtering

Servlet filtering is an integral part of the Servlet 2.3 API. Servlet filtering provides a new type of object called a *filter* that can transform a request or modify a response.

You can chain filters together so that a group of filters can act on the input and output of a specified resource or group of resources.

Filters typically include logging filters, image conversion filters, encryption filters, and Multipurpose Internet Mail Extensions (MIME) type filters (functionally equivalent to the servlet chaining). Although filters are not servlets, their lifecycle is very similar.

Filters are handled in the following manner:

- The Web container determines whether it needs to construct a FilterChain containing the LoggingFilter for the requested resource.

The FilterChain begins with the invocation of the LoggingFilter and ends with the invocation of the requested resource.

- If other filters need to go in the chain, the Web container places them after the LoggingFilter and before the requested resource.

- The Web container then instantiates and initializes the `LoggingFilter` (if it was not done previously) and invokes its `doFilter(FilterConfig)` method to start the chain.
- The `LoggingFilter` preprocesses the request and response objects and then invokes the filter chain `doFilter(ServletRequest, ServletResponse)` method. This method passes the processing to the next resource in the chain (in this case, the requested resource).
- Upon return from the filter chain `doFilter(ServletRequest, ServletResponse)` method, the `LoggingFilter` performs post-processing on the request and response object before sending the response back to the client.

Filter, FilterChain, FilterConfig classes for servlet filtering

The following interfaces are defined as part of the `javax.servlet` package:

- `Filter` interface - methods: `doFilter()`, `getFilterConfig()`, `setFilterConfig()`
- `FilterChain` interface - methods: `doFilter()`
- `FilterConfig` interface - methods: `getFilterName()`, `getInitParameter()`, `getInitParameterNames()`, `getServletContext()`

The following classes are defined as part of the `javax.servlet.http` package:

- `HttpServletRequestWrapper` - methods: See the Servlet 2.3 Specification
- `HttpServletResponseWrapper` - methods: See the Servlet 2.3 Specification

Example: `com.ibm.websphere.LoggingFilter.java`

The following example shows how to implement a filter:

```
package com.ibm.websphere;

import java.io.*;
import javax.servlet.*;

public class LoggingFilter implements Filter
{
    File _loggingFile = null;

    // implement the required init method
    public void init(FilterConfig fc)
    {
        // create the logging file
        xxx;
    }

    // implement the required doFilter method...this is where most of the work is done
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
    {
        try
        {
            // add request info to the log file
            synchronized(_loggingFile)
            {
                xxx;
            }

            // pass the request on to the next resource in the chain
            chain.doFilter(request, response);
        }
        catch (Throwable t)
        {
            // handle problem...
        }
    }

    // implement the required destroy method
}
```

```

public void destroy()
{
    // make sure logging file is closed
    _loggingFile.close();
}
}

```

Configuring page list servlet client configurations

You can define `PageListServlet` configuration information in the IBM Web Extensions file. The IBM Web Extensions file is created and stored in the Web Applications archive (WAR) file by the IBM WebSphere Application Assembly Tool (AAT)

To configure and implement page lists:

1. Use the *PageList Extensions* tab in the Application Assembly Tool (AAT) to configure page list information.
2. Add the `callPage()` method to your servlet to invoke a JavaServer Page (JSP) file in response to a client request.

The `PageListServlet` has a `callPage()` method that invokes a JSP file in response to the HTTP request for a page in a page list. The `callPage()` method can be invoked in one of the following ways:

- `callPage(String pageName, HttpServletRequest request, HttpServletResponse response)`

where the method arguments are:

- `pageName` - a page name defined in the `PageListServlet` configuration
- `request` - the `HttpServletRequest` object
- `response` - the `HttpServletResponse` object
- `callPage(String mlName, String pageName, HttpServletRequest request, HttpServletResponse response)`

where the method arguments are:

- `mlName` - a markup language type
- `pageName` - a page name defined in the `PageListServlet` configuration
- `request` - the `HttpServletRequest` object
- `response` - the `HttpServletResponse` object

3. Use the `PageListServlet` client type detection support to determine the markup language type a calling client requires for the response.

Page lists

Page lists allow you to avoid hard-coding URLs in servlets and JSP files. A page list specifies the location where a request is to be forwarded, but automatically customizes that location depending on the MIME type of the servlet. Use these properties to specify a markup language and an associated MIME type. For the given MIME type, you also specify a set of pages to invoke.

WebSphere Application Server supplies the `PageListServlet` servlet, which you can use to call a JavaServer Pages (JSP) file by name based on the configuration data in the `client_types.xml` file. This file maps a JSP file to a Uniform Resource Identifier (URI). When the URI is invoked, it specifies another JSP file in a Web module. This support allows you to access multiple Uniform Resource Locators (URLs) without hard-coding them in your servlets.

You can also logically group page lists according to the markup language type, such as, Hypertext Markup Language (HTML) or Wireless Markup Language (WML). This allows applications that use servlets to extend the `PageListServlet` servlet, to call JSP files which return the proper markup-language type for the

client request. For example, a request that originates from a PDA device requires WML data. The application server sends the request to a servlet that extends the PageListServlet servlet, and the servlet calls a JSP file that returns a WML response.

Client type detection support

In addition to providing the page list mapping capability, the PageListServlet also provides *Client Type Detection* support. A servlet determines the markup language type that a calling client needs in the response, using the configuration information in the `client_types.xml` file.

Client type detection support allows a servlet, extending the PageListServlet, to call an appropriate JavaServer Pages (JSP) file. The servlet invokes the `callPage()` method, which calls a JSP file based on the markup-language type of the request.

client_types.xml

The `client_types.xml` file provides client type detection support for servlets extending PageListServlet. Using the configuration data in the `client_types.xml` file, servlets can determine the language type that calling clients require for the response.

The client type detection support allows servlets to call appropriate JavaServer Pages (JSP) files with the `callPage()` method. Servlets select JSP files based on the markup-language type of the request.

Servlets must use the following version of the `callPage()` method to determine the markup language type required by the client:

```
callPage(String mlName, String pageName, HttpServletRequest request, HttpServletResponse response)
```

where the arguments are:

- `mlName` - a markup language type
- `pageName` - a page name defined in the PageListServlet configuration
- `request` - the `HttpServletRequest` object
- `response` - the `HttpServletResponse` object

Review the Extending PageListServlet code example to see how the `callPage()` method is invoked by a servlet.

In the example, the client type detection method, `getMLTypeFromRequest(HttpServletRequest request)`, provided by the PageListServlet, inspects the `HttpServletRequest` object request headers, and searches for a match in the `client_types.xml` file.

The client type detection method does the following:

- Uses the input `HttpServletRequest` and the `client_types.xml` file, to check for a matching HTTP request name and value.
- Returns the markup-language value configured for the `<client-type>` element, if a match is found.

If multiple matches are found, this method returns the markup-language for the first `<client-type>` element for which a match is found.

- If no match is found, returns the value of the markup-language for the default page defined in the PageListServlet configuration.

Location

The `client_types.xml` file is located in the `install_root/properties` directory.

Usage notes

- Is this file read-only?

No

- Is this file updated by a product component?

No

- If so, what triggers its update?

This file is created and updated manually by users.

- How and when are the contents of this file used?

Servlets, extending PageListServlet, use this file to determine the language type that calling clients require for the response.

Sample file entry

```
<?xml version="1.0" >
<!DOCTYPE clients [
<!ELEMENT client-type (description, markup-language,request-header+)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT markup-language (#PCDATA)>
<!ELEMENT request-header (name, value)>
<!ELEMENT clients (client-type+)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT value (#PCDATA)>]>
<clients>
  <client-type>
    <description>IBM Speech Client</description>
    <markup-language>VXML</markup-language>
    <request-header>
      <name>user-agent</name>
      <value>IBM VoiceXML pre-release version 000303</value>
    </request-header>
    <request-header>
      <name>accept</name>
      <value>text/vxml</value>
    </request-header>
  </client-type>
  <client-type>
    <description>WML Browser</description>
    <markup-language>WML</markup-language>
    <request-header>
      <name>accept</name>
      <value>text/x-wap.wml</value>
    </request-header>
    <request-header>
      <name>accept</name>
      <value>text/vnd.wap.xml</value>
    </request-header>
  </client-type>
</clients>
```

Example: Extending PageListServlet

The following example shows how a servlet extends the PageListServlet class and determines the markup-language type required by the client. The servlet then uses the callPage() method to call an appropriate JavaServer Pages (JSP) file. In this example, the JSP file that provides the the correct markup-language for the response is *Hello.page*.

```
public class HelloPervasiveServlet extends PageListServlet implements Serializable
{
  /*
  * doGet -- Process incoming HTTP GET requests
  */
  public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws IOException, ServletException
  {
```

```

// This is the name of the page to be called:
String pageName = "Hello.page";

// First check if the servlet was invoked with a queryString that contains a markup-language value.
// For example, if this is how the servlet is invoked:
// http://localhost/servlets/HeloPervasive?mlname=VXML
// then use the following method:
String mlname= getMLNameFromRequest(request);

// If no markup language type is provided in the queryString, then try to determine the client
// type from the request, and use the markup-language name configured in the client_types.xml file.
if (mlName == null)
{
    mlName = getMLTypeFromRequest(request);
}
try
{
    // Serve the request page.
    callPage(mlName, pageName, request, response);
}
catch (Exception e)
{
    handleError(mlName, request, response, e);
}
}
}

```

autoRequestEncoding and autoResponseEncoding

Two new WebSphere Application Server extensions are available in Version 5, `autoRequestEncoding` and `autoResponseEncoding`.

In WebSphere Application Server Version 5, the Web container no longer automatically sets request and response encodings, and response content types. Programmers are expected to set these values using available methods in the Servlet 2.3 Specification. If programmers choose not to use the character encoding methods, they can specify the `autoRequestEncoding` and `autoResponseEncoding` extensions, which enable the application server to set the encoding values and content type.

The values of the `autoRequestEncoding` and `autoResponseEncoding` extensions are either `true` or `false`. The default value for both extensions is `false`. If the value is `false` for both `autoRequestEncoding` and `autoResponseEncoding`, then the request and response character encoding is set to the Servlet 2.3 Specification default, which is ISO-8859-1. Also, If the value is set to `false` for a response, the Web container cannot set a response content type.

Use the Application Assembly Tool (AAT) to change the default values for the `autoRequestEncoding` and `autoResponseEncoding` extensions.

Review the `autoRequestEncoding` and `autoResponseEncoding` encoding examples for a description of Web container behavior when these values are set to `true`.

Examples: autoRequestEncoding and autoResponseEncoding encoding examples

The default value of the `autoRequestEncoding` and `autoResponseEncoding` extensions is `false`, which means that both the request and response character encoding is set to the Servlet 2.3 Specification default of ISO-8859-1. Different character encodings are possible if the client defines character encoding in the

request header, or if the code includes the `setCharacterEncoding(String encoding)` method. Also, If the value is set to false for a response, the Web container cannot set a response content type.

If the `autoRequestEncoding` value is set to true, and the client did not specify character encoding in the request header, and the code does not include the `setCharacterEncoding(String encoding)` method, the Web container tries to determine the correct character encoding for the request parameters and data.

The Web container performs each step in the following list until a match is found:

- Looks at the character set (charset) in the *Content-Type* header.
- Attempts to map the servers locale to a character set using defined properties.
- Attempts to use the `DEFAULT_CLIENT_ENCODING` system property, if one is set.
- Uses the ISO-8859-1 character encoding as the default.

If the `autoResponseEncoding` value is set to true, and the client did not specify character encoding in the request header, and the code does not include the `setCharacterEncoding(String encoding)` method, the Web container does the following:

- Attempts to determine the response content type and character encoding from information in the request header.
- Uses the ISO-8859-1 character encoding as the default.

JavaServer Pages files

JavaServer Pages (JSP) files are application components coded to the Sun Microsystems JavaServer Pages (JSP) Specification. JSP files enable the separation of the Hypertext Markup Language (HTML) code from the business logic in Web pages so that HTML programmers and Java programmers can more easily collaborate in creating and maintaining pages.

The IBM extensions to the JSP Specification include JSP tags that resemble HTML tags. These JSP tags make it easy for HTML authors to add the power of Java technology to Web pages, without being experts in Java programming.

JSP files support a division of roles:

HTML authors

Develop JSP files that access databases and reusable Java components, such as servlets and beans.

Java programmers

Create the reusable Java components and provide the HTML authors with the component names and attributes.

Database administrators

Provide the HTML authors with the name of the database access and table information.

Developing JavaServer Pages files with WebSphere extensions

Several IBM WebSphere extensions are provided for enhancing your JavaServer Pages (JSP) files. This task provides a summary of the extensions that you can utilize.

1. Review the supported specifications.

Create Java components, referring to the JSP specifications from Sun Microsystems.

See Resources for learning for links to coding specifications and examples.

WebSphere Application Server Version 3.5 added IBM extensions to the base Application Programming Interfaces (APIs). Since the JavaServer Pages (JSP) 1.1 and JSP 1.2 Specifications are backward compatible to the JSP 1.0 Specifications, you can invoke the APIs with the IBM extensions without modification.

The extensions belong to these categories:

Syntax for variable data

Put variable fields in JSP files and have servlets and beans dynamically replace the variables with values from a database when the JSP output is returned to the browser.

Syntax for database access

Add a database connection to a Web page and then use that connection to query or update the database. You can provide the user ID and password for the database connection at request time, or you can hard code the user ID and password within the JSP file.

2. Use your favorite integrated development environment (IDE), or a text editor, to develop or migrate code artifacts that meet the specifications.
3. Test the code artifacts.
4. **(Optional)** Batch compile your JSP files if necessary.

Tag libraries

Java ServerPages (JSP) tag libraries contain classes for common tasks such as processing forms and accessing databases from JSP files.

Tag libraries encapsulate, as simple tags, core functionality common to many Web applications. The Java Standard Tag Library (JSTL) supports common programming tasks such as iteration and conditional processing, and provides tags for:

- manipulating XML documents
- supporting internationalization
- using Structured Query Language (SQL)

Tag libraries also introduce the concept of an expression language to simplify page development, and include a version of the JSP expression language.

A tag library has two parts - a Tag Library Descriptor (TLD) file and a JAR file.

tsx:dbconnect tag JavaServer Pages syntax

Use the <tsx:dbconnect> tag to specify information needed to make a connection to a Java Database Connectivity (JDBC) or an Open Database Connectivity (ODBC) database.

The <tsx:dbconnect> syntax does not establish the connection. Use the <tsx:dbquery> and <tsx:dbmodify> syntax instead to reference a <tsx:dbconnect> tag in the same JavaServer Pages (JSP) file to establish the connection.

When the JSP file compiles into a servlet, the Java processor adds the Java coding for the <tsx:dbconnect> syntax to the servlet service() method, which means a new database connection is created for each request for the JSP file.

This section describes the syntax of the <tsx:dbconnect> tag.

```

<tsx:dbconnect id="connection_id"
  userid="db_user" passwd="user_password"
  url="jdbc:subprotocol:database"
  driver="database_driver_name"
  jndiname="JNDI_context/logical_name">
</tsx:dbconnect>

```

where:

- **id**
Represents a required identifier. The scope is the JSP file. This identifier is referenced by the connection attribute of a <tsx:dbquery> tag.
- **userid**
Represents an optional attribute that specifies a valid user ID for the database that you want to access. Specify this attribute to add the attribute and its value to the request object.
Although the userid attribute is optional, you must provide the user ID. See <tsx:userid> and <tsx:passwd> for an alternative to hard coding this information in the JSP file.
- **passwd**
Represents an optional attribute that specifies the user password for the userid attribute. (This attribute is not optional if the userid attribute is specified.) If you specify this attribute, the attribute and its value are added to the request object.
Although the passwd attribute is optional, you must provide the password. See <tsx:userid> and <tsx:passwd> for an alternative to hard coding this attribute in the JSP file.
- **url and driver**
Represents a required attribute if you want to establish a database connection. You must provide the URL and driver.
The application server supports connection to JDBC databases and ODBC databases.
 - For a JDBC database, the URL consists of the following colon-separated elements: jdbc, the subprotocol name, and the name of the database to access. An example for a connection to the Sample database included with IBM DB2 is:


```
url="jdbc:db2:sample"
driver="COM.ibm.db2.jdbc.app.DB2Driver"
```
 - For an ODBC database, use the Sun JDBC-to-ODBC bridge driver included in their Java2 Software Developers Kit (SDK) or another vendor's ODBC driver. The url attribute specifies the location of the database. The driver attribute specifies the name of the driver to use in establishing the database connection.
If the database is an ODBC database, you can use an ODBC driver or the Sun JDBC-to-ODBC bridge. If you want to use an ODBC driver, refer to the driver documentation for instructions on specifying the database location with the url attribute and the driver name.
If you use the bridge, the url syntax is jdbc:odbc:database. An example follows:


```
url="jdbc:odbc:autos"
driver="sun.jdbc.odbc.JdbcOdbcDriver"
```

Note: To enable the application server to access the ODBC database, use the ODBC Data Source Administrator to add the ODBC data source to the System DSN configuration. To access the ODBC Administrator, click the ODBC icon on the Windows NT Control Panel.

- **jndiname**
Represents an optional attribute that identifies a valid context in the application server Java Naming and Directory Interface (JNDI) naming context and the logical name of the data source in that context. The Web administrator configures the context using an administrative client such as the WebSphere Administrative Console.
If you specify the `jndiname` attribute, the JSP processor ignores the `driver` and `url` attributes on the `<tsx:dbconnect>` tag.

An empty element (such as `<url></url>`) is valid.

dbquery tag JavaServer Pages syntax

Use the `<tsx:dbquery>` tag to establish a connection to a database, submit database queries, and return the results set.

The `<tsx:dbquery>` tag does the following:

1. References a `<tsx:dbconnect>` tag in the same JavaServer Pages (JSP) file and uses the information the tag provides to determine the database URL and driver. You can also obtain the user ID and password from the `<tsx:dbconnect>` tag if those values are provided in the `<tsx:dbconnect>` tag.
2. Establishes a new connection
3. Retrieves and caches data in the results object.
4. Closes the connection and releases the connection resource.

This section describes the syntax of the `<tsx:dbquery>` tag.

```
<%-- SELECT commands and (optional) JSP syntax can be placed within the tsx:dbquery. --%>
<%-- Any other syntax, including HTML comments, are not valid. --%>
<tsx:dbquery id="query_id" connection="connection_id" limit="value" >
</tsx:dbquery>
```

where:

- **id**
Represents the identifier of this query. The scope is the JSP file. Use `id` to reference the query. For example, from the `<tsx:getProperty>` tag, use `id` to display the query results.
The `id` becomes the name of a bean that contains the results set. The bean properties are dynamic and the property names are the names of the columns in the results set. If you want different column names, use the SQL keyword for specifying an alias on the `SELECT` command. In the following example, the database table contains columns named `FNAME` and `LNAME`, but the `SELECT` statement uses the `AS` keyword to map those column names to `FirstName` and `LastName` in the results set:
`Select FNAME, LNAME AS FirstName, LastName from Employee where FNAME='Jim'`
- **connection**
Represents the identifier of a `<tsx:dbconnect>` tag in this JSP file. The `<tsx:dbconnect>` tag provides the database URL, driver name, and optionally, the user ID and password for the connection.
- **limit**
Represents an optional attribute that constrains the maximum number of records returned by a query. If this attribute is not specified, no limit is used. In such a case, the effective limit is determined by the number of records and the system caching capability.
- `SELECT` command and JSP syntax

Represents the only valid SQL command, SELECT. The <tsx:dbquery> tag must return a results set. Refer to your database documentation for information about the SELECT command. See other articles in this section for a description of JSP syntax for variable data and inline Java code.

dbmodify tag JavaServer Pages syntax

The <tsx:dbmodify> tag establishes a connection to a database and then adds records to a database table.

The <tsx:dbmodify> tag does the following:

1. References a <tsx:dbconnect> tag in the same JavaServer Pages (JSP) file and uses the information provided by that tag to determine the database URL and driver.

Note: You can also obtain the user ID and password from the <tsx:dbconnect> tag if those values are provided in the <tsx:dbconnect> tag.

2. Establishes a new connection.
3. Updates a table in the database.
4. Closes the connection and releases the connection resource.

This section describes the syntax of the <tsx:dbmodify> tag.

```
<%-- Any valid database update commands can be placed within the DBMODIFY tag. -->
<%-- Any other syntax, including HTML comments, are not valid. -->
<tsx:dbmodify connection="connection_id">
</tsx:dbmodify>
```

where:

- **connection**

Represents the identifier of a <tsx:dbconnect> tag in this JSP file. The <tsx:dbconnect> tag provides the database URL, driver name, and (optionally) the user ID and password for the connection.

- Database commands

Represents valid database commands. Refer to your database documentation for details

tsx:getProperty tag JavaServer Pages syntax and examples

The <tsx:getProperty> tag gets the value of a bean to display in a JavaServer Pages (JSP) file.

This IBM extension of the Sun JSP <jsp:getProperty> tag implements all of the <jsp:getProperty> function and adds the ability to introspect a database bean created using the IBM extension <tsx:dbquery> or <tsx:dbmodify>.

Note: You cannot assign the value from this tag to a variable. The value, generated as output from this tag, displays in the browser window.

This section describes the syntax of the <tsx:getProperty> tag:

```
<tsx:getProperty name="bean_name"
  property="property_name" />
```

where:

- **name**

Represents the name of the bean declared by the id attribute of a <tsx:dbquery> syntax within the JSP file. See <tsx:dbquery> for an explanation. The value of this attribute is case-sensitive.

- **property**

Represents the property of the bean to access for substitution. The value of the attribute is case-sensitive and is the locale-independent name of the property.

Tag example:

```
<tsx:getProperty name="userProfile" property="username" />
<tsx:getProperty name="request" property=request.getParameter("corporation") />
```

In most cases, the value of the property attribute is just the property name. However, to access the request bean or to access a property of a property (sub property), specify the full form of the property attribute. The full form also gives you the option to specify an index for indexed properties. You can specify the optional index as a constant (such as 2), or an index like the one described in the <tsx:repeat> tag. Some examples using the full form of the property attribute follow:

```
<tsx:getProperty name="staffQuery" property=address(currentAddressIndex) />
<tsx:getProperty name="shoppingCart" property=items(4).price />
<tsx:getProperty name="fooBean" property=foo(2).bat(3).boo.far />
```

tsx:userid and tsx:passwd tag JavaServer Pages syntax

With the <tsx:userid> and <tsx:passwd> tags, you do not have to hard code a user ID and password in the <tsx:dbconnect> tag.

Use the <tsx:userid> and <tsx:passwd> tags to accept user input for the values and then add that data to the request object. You can access the request object with a JavaServer Pages (JSP) file, such as the *JSPEmployee.jsp* example that requests the database connection.

You must use <tsx:userid> and <tsx:passwd> tags within a <tsx:dbconnect> tag.

This section describes the syntax of the <tsx:userid> and <tsx:passwd> tags.

```
<tsx:dbconnect id="connection_id"
  <font color="red"><userid></font><tsx:getProperty name="request" property=request.getParameter("userid") /><font color="red"><passwd></font><tsx:getProperty name="request" property=request.getParameter("passwd") /><font color="red">url="protocol:database_name:database_table"
  driver="JDBC_driver_name">
</tsx:dbconnect>
```

where:

- **<tsx:getProperty>**

Represents the syntax as a mechanism for embedding variable data.

- **userid**

Represents a reference to the request parameter that contains the user ID. You must add the parameter to the request object that passes to this JSP file. You can set the attribute and its value in the request object, using an HTML form or a URL query string to pass the user-specified request parameters.

- **passwd**

Represents a reference to the request parameter that contains the password. Add the parameter to the request object that passes to this JSP file. You can set the attribute and its value in the request object, using an HTML form or a URL query string, to pass user-specified values.

tsx:repeat tag JavaServer Pages syntax

The <tsx:getProperty> tag repeats a block of HTML tagging.

Use the `<tsx:repeat>` syntax to iterate over a database query results set. The `<tsx:repeat>` syntax iterates from the start value to the end value until one of the following conditions is met:

- The end value is reached.
- An exception is thrown.

The output of a `<tsx:repeat>` block is buffered until the block completes. If an exception is thrown before a block completes, no output is written for that block.

This section describes the syntax of the `<tsx:repeat>` tag:

```
<tsx:repeat index=name start="starting_index" end="ending_index">
</tsx:repeat>
```

where:

- **index**

Represents an optional name used to identify the index of this repeat block. The value is case-sensitive and its scope is the JSP file.

- **start**

Represents an optional starting index value for this repeat block. The default is 0.

- **end**

Represents an optional ending index value for this repeat block. The maximum value is 2,147,483,647.

If the value of the end attribute is less than the value of the start attribute, the end attribute is ignored.

Example: Combining `tsx:repeat` and `tsx:getProperty` JavaServer Pages tags

The following code snippet shows you how to code these tags:

```
<tsx:repeat>
<tr>
  <td><tsx:getProperty name="empqs" property="EMPNO" />
  <tsx:getProperty name="empqs" property="FIRSTNME" />
  <tsx:getProperty name="empqs" property="WORKDEPT" />
  <tsx:getProperty name="empqs" property="EDLEVEL" />
</td>
</tr>
</tsx:repeat>
```

Example: `tsx:dbmodify` tag syntax

In the following example, a new employee record is added to a database. The values of the fields are based on user input from this JavaServer Pages (JSP) file and referenced in the database commands using the `<tsx:getProperty>` tag.

```
<tsx:dbmodify connection="conn" >
insert into EMPLOYEE
  (EMPNO,FIRSTNME,MIDINIT,LASTNAME,WORKDEPT,EDLEVEL)
values
('<tsx:getProperty name="request" property=request.getParameter("EMPNO") />',
'<tsx:getProperty name="request" property=request.getParameter("FIRSTNME") />',
'<tsx:getProperty name="request" property=request.getParameter("MIDINIT") />',
'<tsx:getProperty name="request" property=request.getParameter("LASTNAME") />',
'<tsx:getProperty name="request" property=request.getParameter("WORKDEPT") />',
<tsx:getProperty name="request" property=request.getParameter("EDLEVEL") />)
</tsx:dbmodify>
```

Example: Using `tsx:repeat` JavaServer Pages tag to iterate over a results set

The `<tsx:repeat>` tag iterates over a results set. The results set is contained within a bean. The bean can be a static bean, for example, a bean created by using the IBM WebSphere Studio database wizard, or a dynamically generated bean, for example, a bean generated by the `<tsx:dbquery>` syntax. The following table is a graphic representation of the contents of a bean called, *myBean*:

	col1	col2	col3
row0	friends	Romans	countrymen
row1	bacon	lettuce	tomato
row2	May	June	July

Some observations about the bean:

- The column names in the database table become the property names of the bean. The `<tsx:dbquery>` section describes a technique for mapping the column names to different property names.
- The bean properties are indexed. For example, `myBean.get(Col1(row2))` returns May.
- The query results are in the rows. The `<tsx:repeat>` tag iterates over the rows, beginning at the start row.

The following table compares using the `<tsx:repeat>` tag to iterate over a static bean, versus a dynamically generated bean:

Static Bean Example	<code><tsx:repeat></code> Bean Example
<p>myBean.class</p> <pre>// Code to get a connection // Code to get the data Select * from myTable; // Code to close the connection</pre> <p>JSP file</p> <pre><tsx:repeat index=abc> <tsx:getProperty name="myBean" property="col1(abc)" /> </tsx:repeat></pre> <p>Notes:</p> <ul style="list-style-type: none"> • The bean (<code>myBean.class</code>) is a static bean. • The method to access the bean properties is <code>myBean.get(property(index))</code>. • You can omit the property index, in which case the index of the enclosing <code><tsx:repeat></code> tag is used. You can also omit the index on the <code><tsx:repeat></code> tag. • The <code><tsx:repeat></code> tag iterates over the bean properties row by row, beginning with the start row. 	<p>JSP file</p> <pre><tsx:dbconnect id="conn" userid="alice"passwd="test" url="jdbc:db2:sample" driver="COM.ibm.db2.jdbc.app.DB2Driver"> </tsx:dbconnect > <tsx:dbquery id="dynamic" connection="conn" > Select * from myTable; </tsx:dbquery> <tsx:repeat index=abc> <tsx:getProperty name="dynamic" property="col1(abc)" /> </tsx:repeat></pre> <p>Notes:</p> <ul style="list-style-type: none"> • The bean (dynamic) is generated by the <code><tsx:dbquery></code> tag and does not exist until the syntax executes. • The method to access the bean properties is <code>dynamic.getValue("property", index)</code>. • You can omit the property index, in which case the index of the enclosing <code><tsx:repeat></code> tag is used. You can also omit the index on the <code><tsx:repeat></code> tag. • The <code><tsx:repeat></code> tag syntax iterates over the bean properties row by row, beginning with the start row.

Implicit and explicit indexing

Examples 1, 2, and 3 show how to use the `<tsx:repeat>` tag. The examples produce the same output if all indexed properties have 300 or fewer elements. If there are more than 300 elements, Examples 1 and 2 display all elements, while Example 3 shows only the first 300 elements.

Example 1 shows *implicit indexing* with the default start and default end index. The bean with the smallest number of indexed properties restricts the number of times the loop repeats.

```
<table>
<tsx:repeat>
  <tr><td><tsx:getProperty name="serviceLocationsQuery" property="city" /></tr></td>
  <tr><td><tsx:getProperty name="serviceLocationsQuery" property="address" /></tr></td>
  <tr><td><tsx:getProperty name="serviceLocationsQuery" property="telephone" /></tr></td>
</tsx:repeat>
</table>
```

Example 2 shows indexing, starting index, and ending index:

```
<table>
<tsx:repeat index=myIndex start=0 end=2147483647>
  <tr><td><tsx:getProperty name="serviceLocationsQuery" property=city(myIndex) /></tr></td>
  <tr><td><tsx:getProperty name="serviceLocationsQuery" property=address(myIndex) /></tr></td>
  <tr><td><tsx:getProperty name="serviceLocationsQuery" property=telephone(myIndex) /></tr></td>
</tsx:repeat>
</table>
```

Example 3 shows *explicit indexing* and ending index with implicit starting index. Although the index attribute is specified, you can still implicitly index the indexed property city because the (myIndex) tag is not required.

```
<table>
<tsx:repeat index=myIndex end=299>
  <tr><td><tsx:getProperty name="serviceLocationsQuery" property="city" /></tr></td>
  <tr><td><tsx:getProperty name="serviceLocationsQuery" property="address(myIndex)" /></tr></td>
  <tr><td><tsx:getProperty name="serviceLocationsQuery" property="telephone(myIndex)" /></tr></td>
</tsx:repeat>
</table>
```

Nesting `<tsx:repeat>` blocks

You can nest `<tsx:repeat>` blocks. Each block is separately indexed. This capability is useful for interleaving properties on two beans, or properties that have subproperties. In the example, two `<tsx:repeat>` blocks are nested to display the list of songs on each compact disc in the user's shopping cart.

```
<tsx:repeat index=cdindex>
  <h1><tsx:getProperty name="shoppingCart" property=cds.title /></h1>
  <table>
  <tsx:repeat>
    <tr><td><tsx:getProperty name="shoppingCart" property=cds(cdindex).playlist />
    </td></tr>
  </tsx:repeat>
  </table>
</tsx:repeat>
```

JspBatchCompiler tool

As an IBM enhancement to JavaServer Pages support, IBM WebSphere Application Server provides a batch JSP compiler. Use this function to batch compile your JSP files and thereby enable faster responses to the initial client requests for the JSP files on your production Web server.

Batch compiling makes the first request for a JSP file much faster because the JSP file is translated and compiled into a servlet. Batch compiling is also useful as a fast way to resynchronize all of the JSP files for an application.

To use the JSP batch compiler for JSP files, enter the following command on a single line at an operating system command prompt:

```
JspBatchCompiler -enterpriseapp.name <name>
                  [-webmodule.name <name>]
                  [-cell.name <name>]
                  [-node.name <name>]
                  [-server.name <name>]
                  [-filename <jsp name>]
                  [-keepgenerated <true|false>]
                  [-verbose <true|false>]
                  [-deprecation <true|false>]
```

If the names specified for these arguments are comprised of two or more words separated by spaces, you must add quotation marks around the names.

where:

- **enterpriseapp.name**
Represents the name of the enterprise application you want to compile.
- **webmodule.name**
Represents the name of the specific Web module that you want to compile. If this argument is not set, all Web modules in the enterprise application are compiled.
- **cell.name**
Represents the name of the cell in which the application is deployed. The default is `BaseApplicationServerCell`.
- **node.name**
Represents the name of the node in which the application is deployed. The default is `DefaultNode`.
- **server.name**
Represents the name of the server in which the application is deployed. The default is `server1`.
- **filename**
Represents the name of a single JSP file that you want to compile. If this argument is not set, all files in the Web module are compiled. Alternatively, if *filename* is set to the name of a directory, only the JSP files in that directory are compiled.
- **keepgenerated**
Represents the option to save or erase the generated files.
If set to yes, WebSphere Application Server saves the generated `.java` files used for compilation on your server. By default, this argument is set to no and the `.java` files are erased after the class files have compiled.
- **verbose**
Indicates the compiler should generate verbose output while compiling the generated sources.
- **deprecation**
Indicates the compiler should generate deprecation warnings while compiling the generated sources.

Bean Scripting Framework

The Bean Scripting Framework (BSF) enables you to use scripting language functions in your Java server-side applications. This framework also extends scripting languages so that you can use existing Java classes and Java beans in the JavaScript language.

With BSF, you can write scripts that create, manipulate and access values from Java objects, or you can write Java programs that evaluate and access results from scripts.

WebSphere Application Server provides the Bean Scripting Framework, which consists of a BSF manager, a BSF engine, and a scripting engine.

BSF provides an access mechanism to Java objects for the scripting languages it supports, so that both the scripting language and the Java code can access code exclusive functions. The access mechanism is implemented through a registry of objects maintained by BSF.

BSF in WebSphere Application Server supports the Rhino ECMAScript.

The "Resources for Learning" article provides external BSF links that document future supported languages.

Developing Web applications

Design a Web application and the components that it needs.

For general Web application design information, see "Resources for learning."

There are two basic approaches to selecting tools for developing Web applications:

- You can use one of the available integrated development environments (IDEs). IDE tools automatically generate significant parts of the servlet and JavaServer Pages (JSP) code, and Hypertext Markup Language (HTML) files. They also contain integrated tools for packaging and testing the Web application components. The IBM WebSphere Application Developer product is the recommended IDE. For more information, see the documentation for that product.
- If you decide to develop Web components without an IDE, you need at least an ASCII text editor. You can also use tools available in the Java Software Development Kit (SDK) and in this product to assemble, test, and deploy the Web application components.

The following steps support the second approach, development without an IDE.

1. If necessary, migrate any pre-existing code to the required version of the servlet and JSP specification.
2. Write and compile the components of the Web application. To access classes that were extended, compile your code using the `-classpath` option on the `javac` compiler. This option allows you to reference the `j2ee.jar` file in the product `<install_root>\lib` directory.

For example, to compile a servlet running on the Windows NT version of WebSphere Application Server, specify:

```
javac -classpath D:\Program Files\WebSphere\AppServer\lib\j2ee.jar MyServlet.java
```

To compile that same servlet on the Windows NT version of WebSphere Network Deployment, specify:

```
javac -classpath D:\Program Files\WebSphere\DeploymentManager\lib\j2ee.jar MyServlet.java
```

3. **5.0.1 +** Optional: Disable JavaServer Pages (JSP) runtime compilation, if necessary.

Assemble the application components in one or more Web modules.

Disabling JavaServer Pages run-time compilation

By default, the JavaServer Pages (JSP) engine translates a requested JSP file, compiles the .java file, and loads the compiled servlet into the run-time environment. In previous releases of WebSphere Application Server, if a .class file did not exist, the JSP engine always translated and compiled the JSP file. You had to turn off the Web applications reload capability to prevent additional translations and recompiles of the file.

With Version 5.0.1 of WebSphere Application Server, you can now change the JSP engine default behavior by indicating a JSP file should never be translated or compiled at run time, even when a .class file does not exist.

If run-time compilation is disabled, you must precompile the JSP files, which provides the following advantages:

- Reduces compilation related disk operations.
- Minimizes disk storage requirements necessary for handling temporary .java and .class files generated during a run-time compilation.
- Forces you to verify that a JSP file compiled successfully before deploying and installing the application in WebSphere Application Server.

You can disable run-time JSP file compilation on a global or an individual Web application basis:

- To disable the translation and compilation of JSP files for all Web applications, set the Web container Custom property `disableJspRuntimeCompilation` to true.

Set this property through the Web container Custom properties panel in the administrative console. To view this administrative console page, click **Servers > Application Servers > server_name > Web Container > Custom Properties > property_name**.

Valid values for this setting are true or false. If this property is set to true, then translation and compilation of the JSP files is disabled at run time for all Web applications.

- To disable the translation and compilation of JSP files for a specific Web application, set the JSP engine initialization parameter `disableJspRuntimeCompilation` to true. This setting, if enabled, determines the run-time behavior of the JSP engine and overrides the Web container custom property setting.

Set this parameter through the JavaServer Pages attribute assembly settings panel in the Application Assembly Tool (AAT). To view this page, click:

Web Modules > component_instance > Assembly Property Extensions

Valid values for this setting are true or false. If this parameter is set to true, then, for that specific Web application, translation and compilation of the JSP files is disabled at run time, and the JSP engine only loads precompiled files.

- If neither the Web container custom property nor the JSP attribute assembly parameter is set, the first request for a JSP file results in the translation and

compilation of the JSP file when the .class file does not exist. Subsequent requests for the file also result in compilations and translations, but only if the following conditions are met:

- Compilations and translations are required.
- Reloading is enabled for the Web module.
- Reload interval is exceeded.

If you disable run-time compilation and a request arrives for a JSP file that does not have a matching .class file, the JSP engine returns HTTP error 501 (Not implemented) to the browser. If the JSP file does not exist, the JSP engine returns HTTP error 404 (File not found) to the browser. In both cases, an exception is written to the System Out (SYSOUT) and First Failure Data Capture (FFDC) logs. If a JSP file has a matching .class file but that file is out of date, the JSP engine still loads the .class file into memory.

Perform the following steps to determine whether the `disableJspRuntimeCompilation` option is enabled in WebSphere Application Server:

1. Enable the Diagnostic Trace Service and set the trace specification to `com.ibm.ws.webcontainer.jsp.servlet.*=all=enabled`.
2. Request a JSP file.
3. Locate the string, `disableJspRuntimeCompilation:true`, in the `trace.log` file.
4. Ensure the `jspUri:` entry matches the requested JSP file.

If both the `disableJspRuntimeCompilation:true` string and the matching `jspUri:` entry appear in the trace, the `disableJspRuntimeCompilation` setting is enabled for the Web application.

Example: Converting JavaScript source to the Bean Scripting Framework

JavaScript code is one of the most popular languages of Web developers. This language supports the following base objects, plus additional objects from the Document Object Model:

- array
- date
- math
- number
- string

Server-side JavaScript code supports the same base objects, and additional objects that support user access to databases, file systems and e-mail systems.

Like client-side JavaScript code, server-side JavaScript code is also platform, browser, and language independent.

You can convert server-side JavaScript applications to the Bean Scripting Framework. This article describes how to perform this conversion.

Server-side JavaScript source code

Suppose you have the following server-side JavaScript application:

```
<html>
<head>
<title>Hello World server-side JavaScript example</title>
</head>
<body>
```

```

<br><br>
</body>
</html>

<server>
function writePage()
  write("<center><font size='6'>Hello World</font></center>");
</server>

```

Converting server-side JavaScript source code to the Bean Scripting Framework (BSF)

Make the following changes to the JavaScript source code to enable BSF:

```

<%@ page language="javascript" %>
<html>
<head>
<title>Hello World server-side BSF/JavaScript example</title>
</head>
<body>
<br><br>
</body>
</html>

<%
  out.println("<center><font size='6'>Hello World</font></center>");
%>

```

Review the other BSF reference articles for deployment information and additional programming examples.

Scenario: Creating a Bean Scripting Framework application

Scenario description

Programming skills in JavaScript code are more prevalent than programming skills using JavaServer Pages (JSP) tags. Using the Bean Scripting Framework, JavaScript programmers can gradually introduce JSP tags in their JavaScript applications without completely rewriting the source code. The BSF method not only reduces the potential of programming errors, but also provides a painless way to learn a new technology.

The following scenario illustrates how to implement a BSF application using JavaScript within JSP tags.

Developing the BSF application

At ABC elementary school, John Doe teaches third grade mathematics. He wants to help his students memorize their multiplication tables, and thinks a small Web-based quiz could help meet his objective. However, John Doe only knows JavaScript.

Using the Bean Scripting Framework to help leverage his JavaScript skills, John Doe creates two JSP files, `multiplication_test.jsp` and `multiplication_scoring.jsp`.

In the `multiplication_test.jsp` file, John Doe uses both client-side and server-side JavaScript code to generate a test of 100 random multiplication questions, displayed using a three minute timer. He then writes the

multiplication_scoring.jsp file to read the data submitted by the multiplication_test.jsp file and to generate the scoring results.

John Doe creates the following two files:

multiplication_test.jsp:

```
<html>
<head>
<title>Multiplication Practice Test</title>
<script language="javascript">
var countMin=3;
var countSec=0;
function updateDisplay (min, sec) {
    var disp;
    if (min <= 9) disp = " 0";
    else disp = " ";
    disp += (min + ":");
    if (sec <= 9) disp += ("0" + sec);
    else disp += sec;
    return(disp);
}
function countDown() {
    countSec--;
    if (countSec == -1) {
        countSec = 59;
        countMin--;
    }
    document.mulptest.counter.value = updateDisplay(countMin, countSec);
    if((countMin == 0) &&(countSec == 0)) document.mulptest.submit();
    else var down = setTimeout("countDown();", 1000);
}
</script>
</head>
<body bgcolor="#ffffff" onLoad="countDown();">
<%@ page language="javascript" %>
<h1>Three Minute Multiplication Drill</h1>
<hr>
<h2>Remember: this is an opportunity to excel!</h2>
<p>
<form method="POST" name="multtest" action="multiplication_scoring.jsp">
<div align="center">
<table>
<tr>
<td>
<h3>Time left:
<input type="text" name="counter" size="9" value="03:00" readonly>
</h3>
</td>
<td>
<input type="submit" value="Submit for scoring!">
</td>
</tr>
</table>
<table border="1">
<%
var newrow = 0;
var q_num = 0;
function addQuestion(num1, num2) {
    if (newrow == 0) out.println("<tr>");
    out.println("<td>");
    out.println(num1 + " x " + num2 + " = ");
    out.println("</td><td>");
    out.print("<input name=\"\" + q_num + \"|\" + num1 + \":\" + num2 + \"\");
    out.println("<type=\"text\" size=\"10\">");
    out.println("</td>");
    if (newrow == 3) {
        out.println("</tr>");
    }
}
```

```

        newrow = 0;
    }
    else newrow++;
    q_num++;
}
for (var i = 0; i < 100; i++) {
    var rand1 = Math.ceil(Math.random() * 12);
    var rand2 = Math.ceil(Math.random() * 12);
    addQuestion(rand1, rand2);
}
}
%>
</table>
</div>
</form>
</body>
</html>

```

multiplication_scoring.jsp:

```

<html>
<head>
<title>Multiplication Practice Test Results</title>
</head>
<body bgcolor="#ffffff">
<%@ page language="javascript" %>
<h1>Multiplication Drill Score</h1>
<hr>
<div align="center">
<table border="1">
<tr><th>Problem</th><th>Correct Answer</th><th>Your Answer</th></tr>
<%
var total_score = 0;
function score (current, pos1, pos2) {
    var multiplier = current.substring(pos1 + 1, pos2);
    var multiplicand = current.substring(pos2 + 1, current.length());
    var your_product = request.getParameterValues(current)[0];
    var true_product = multiplier * multiplicand;
    out.println("<tr>");
    out.println("<td>" + multiplier + " x " + multiplicand + " = </td>");
    out.println("<td>" + true_product + "</td>");
    if (your_product == true_product) {
        total_score++;
        out.print("<td bgcolor=\"\#00ff00\">");
    }
    else {
        out.print("<td bgcolor=\"\#ff0000\">");
    }
    out.println(your_product + "</td>");
    out.println("</tr>");
}
var equations = request.getParameterNames();
while(equations.hasMoreElements()) {
    var currElt = equations.nextElement();
    var splitPos1 = currElt.indexOf("|");
    var splitPos2 = currElt.indexOf(":");
    if (splitPos1 >=0 && splitPos2 >= 0) score(currElt, splitPos1, splitPos2);
}
%>
</table>
<h2>Total Score: <%= total_score %></h2>
<h3><a href="multiplication_test.jsp">Try again?</a></h3>
</div>
</body>
</html>

```

Follow these steps to see how John Doe uses BSF to implement JavaScript in a JSP application:

1. Give your files a .jsp extension.

2. Use server-side JavaScript code in your application.

The `multiplication_test.jsp` file incorporates both client-side and server-side JavaScript. Server-side JavaScript is similar to client-side JavaScript; the primary difference consists of using a different set of objects. Whereas client-side JavaScript programmers invoke `document` and `window` objects, server-side JavaScript programmers, using the Bean Scripting Framework, invoke a set of objects provided by the JSP technology. Also, client-side scripts are enclosed in `<script>` tags, but server-side scripts use JSP scriptlet and expression tags.

Examine the following blocks of code:

```
<script language="javascript">
var countMin=3;
var countSec=0;
function updateDisplay (min, sec) {
    var disp;
    if (min <= 9) disp = " 0";
    else disp = " ";
    disp += (min + ":");
    if (sec <= 9) disp += ("0" + sec);
    else disp += sec;
    return(disp);
}
function countDown() {
    countSec--;
    if (countSec == -1) {
        countSec = 59;
        countMin--;
    }
    document.multtest.counter.value = updateDisplay(countMin, countSec);
    if((countMin == 0) && (countSec == 0)) document.multtest.submit();
    else var down = setTimeout("countDown()", 1000);
}
</script>
....
<body bgcolor="#ffffff" onLoad="countDown();">
...
<form method="POST" name="multtest" action="multiplication_scoring.jsp">
...
<input type="text" name="counter" size="9" value="03:00" readonly>
...
```

The JavaScript code contained in the `<script>` block implements a timer set within the `<input>` field named `counter`. The `onLoad` event handler in the `<body>` tag causes the browser to load and execute the code when the the page is loaded.

The `document.multtest.submit()` statement causes the form named `multtest` to be submitted when the timer expires.

3. Identify the code to the BSF function.

The following code example, from the `multiplication_test.jsp` file, displays the use of a JSP directive. This directive tells the WebSphere Application Server BSF function that this file is using the JavaScript language, and that the JavaScript code is enclosed by the `<% ... %>` scriptlet tags. The out implicit JSP object in this code example, creates the body section of a table from 100 randomly generated questions.

```
...
<%@ page language="javascript" %>
...
<%
var newrow = 0;
var q_num = 0;
```



```

function addQuestion(num1, num2) {
    if (newrow == 0) out.println("<tr>");

    out.println("<td>");
    out.println(num1 + " x " + num2 + " = ");
    out.println("</td><td>");
    out.print("<input name=\"" + q_num + "|" + num1 + ":" + num2 + "\" ");
    out.println("type=\"text\" size=\"10\">");
    out.println("</td>");

    if (newrow == 3) {
        out.println("</tr>");
        newrow = 0;
    }
    else newrow++;

    q_num++;
}

for (var i = 0; i < 100; i++) {
    var rand1 = Math.ceil(Math.random() * 12);
    var rand2 = Math.ceil(Math.random() * 12);

    addQuestion(rand1, rand2);
}

%>
...

```

4. Read the results.

To score the results of the practice drill, John Doe uses the request implicit JSP object in the `multiplication_scoring.jsp` file to obtain the POST data created within the `<form>` tags in the `multiplication_test.jsp` file.

The `multiplication_scoring.jsp` file uses the POST data to build an output file containing the original question, the student's answer, and the correct answer, and then prints the text in a table format using the `out` implicit object.

The following code example from the `multiplication_scoring.jsp` file illustrates the use of the `request` and `out` JSP objects:

```

...
<%@ page language="javascript" %>
...
<%
var total_score = 0;
function score (current, pos1, pos2) {
    var multiplier = current.substring(pos1 + 1, pos2);
    var multiplicand = current.substring(pos2 + 1, current.length());
    var your_product = request.getParameterValues(current)[0];
    var true_product = multiplier * multiplicand;
    out.println("<tr>");
    out.println("<td>" + multiplier + " x " + multiplicand + " = </td>");
    out.println("<td>" + true_product + "</td>");
    if (your_product == true_product) {
        total_score++;
        out.print("<td bgcolor=\"\#00ff00\">");
    }
    else {
        out.print("<td bgcolor=\"\#ff0000\">");
    }
    out.println(your_product + "</td>");
    out.println("</tr>");
}

var equations = request.getParameterNames();
while(equations.hasMoreElements()) {
    var currElt = equations.nextElement();
    var splitPos1 = currElt.indexOf("|");

```

```

        var splitPos2 = currElt.indexOf(":");
        if (splitPos1 >=0 && splitPos2 >= 0) score(currElt, splitPos1, splitPos2);
    }
%>
...
<h2>Total Score: <%= total_score %></h2>
...

```

Note: Although using separate scriptlet blocks of code for different portions of a conditional expression is common in JSP files implemented in Java, it is invalid for JSP files implemented using JavaScript through the Bean Scripting Framework. The JavaScript code must be entirely contained within the scriptlet tags.

The following code example illustrates invalid usage:

```

<% if (pass == 0) %>
    <i>pass is true</i>
<% else %>
    <i>pass is not true</i>

```

Deploying the BSF application

You assemble and deploy BSF applications in the same manner as JSP applications. Review the [Assembling applications](#) article for more information.

Deploy the BSF code examples in WebSphere Application Server to view this applications processing and output. Use the following quick steps to deploy the application.

Note: The intent of these "quick steps" is to provide you with instant application output. However, the supported method for deployment is the same as for standard JSP files.

1. Use the DefaultApplication to add your BSF files.

Copy your .jsp files to the DefaultApplication directory:<app server install directory>/installedApps/<node name>/DefaultApplication.ear/DefaultApplication.war

2. Start the application server.
3. Open a browser and request your BSF application.

Use the following URL to request your application:

<http://hostName:9080/<jspFileName>.jsp>

Example: Bean Scripting Framework code example

The following code examples show how to implement JavaScript using the Bean Scripting Framework (BSF).

For a quick demonstration of the BSF function, copy these code examples into 2 separate files, and deploy them in WebSphere Application Server using the instructions in the [BSF scenario](#) article.

Multiplication practice test

```

<html>
<head>
<title>Multiplication Practice Test</title>
<!--

```

This file and its companion, multiplication_score.jsp, illustrate the

use of ECMAScript within the BSF framework. The task is a simple timed math quiz, which is 3 minutes in duration. When the quiz ends, the score is computed and displayed. Users are then asked if they wish to try the quiz again.

```
-->

<!--
This code fragment displays and updates the quiz
countdown in client side JavaScript code.
-->
<script language="javascript">
var countMin=3;
var countSec=0;

// This code computes the current countdown time.
function updateDisplay (min, sec) {
    var disp;

    if (min <= 9) disp = " 0";
    else disp = " ";

    disp += (min + ":");

    if (sec <= 9) disp += ("0" + sec);
    else disp += sec;

    return(disp);
}

//This code fragment displays the current countdown time in the user's
// browser window, and submits the results for scoring when the countdown
// ends.

function countDown() {
    countSec--;
    if (countSec == -1) {
        countSec = 59;
        countMin--;
    }
    document.mulpttest.counter.value = updateDisplay(countMin, countSec);
    if((countMin == 0) && (countSec == 0)) document.mulpttest.submit();
    else var down = setTimeout("countDown();", 1000);
}

</script>
</head>
<body bgcolor="#ffffff" onLoad="countDown();">

<!--
The body of the quiz runs as JavaServer Pages (JSP) code using BSF.
The code outputs the problems in table format using the POST method
and invokes the scoring module when the user chooses to end the quiz
or when the countdown ends.
-->
<%@ page language="javascript" %>

<h1>Three Minute Multiplication Drill</h1>
<hr>

<h2>Remember: this is an opportunity to excel!</h2>
<p>

<form method="POST" name="multtest" action="multiplication_scoring.jsp">
<div align="center">
<table>
<tr>
<td>
```

```

<h3>Time left:
<input type="text" name="counter" size="9" value="03:00" readonly>
</h3>
</td>
<td>
<input type="submit" value="Submit for scoring!">
</td>
</tr>
</table>
<table border="1">
<%
var newrow = 0;
var q_num = 0;

// This code generates a new random multiplication problem up to
// the number twelve, and enters it into the table of problems.

function addQuestion(num1, num2) {
    if (newrow == 0) out.println("<tr>");

    out.println("<td>");
    out.println(num1 + " x " + num2 + " = ");
    out.println("</td><td>");
    out.print("<input name=\"" + q_num + "\"" + num1 + ":" + num2 + "\" ");
    out.println("type=\"text\" size=\"10\">");
    out.println("</td>");

    if (newrow == 3) {
        out.println("</tr>");
        newrow = 0;
    }
    else newrow++;

    q_num++;
}

//This code obtains two random operands and formats 100 quiz problems.

for (var i = 0; i < 100; i++) {
    var rand1 = Math.ceil(Math.random() * 12);
    var rand2 = Math.ceil(Math.random() * 12);

    addQuestion(rand1, rand2);
}

%>
</table>
</div>
</form>

</body>
</html>

```

Multiplication practice test results

```

<html>
<head>
<title>Multiplication Practice Test Results</title>
</head>
<body bgcolor="#ffffff">

```

```

<!--
This JSP code is invoked when the user submits a math quiz for scoring,
or when the quiz countdown expires. The JSP code tabulates the problem
list, the correct answer, the user's answer, and scores the test. It
then offers the user an opportunity to try the quiz again.

```

```

-->
<%@ page language="javascript" %>

<h1>Multiplication Drill Score</h1>
<hr>

<div align="center">
<table border="1">
<tr><th>Problem</th><th>Correct Answer</th><th>Your Answer</th></tr>
<%
var total_score = 0;

// This code parses the submitted form, extracts the a problem generated by the
// multiplication_test.jsp file, outputs it, computes the correct answer,
// and displays this information and the user answer. The code scores
// the quiz using a running sum of correct answers.

function score (current, pos1, pos2) {
    var multiplier = current.substring(pos1 + 1, pos2);
    var multiplicand = current.substring(pos2 + 1, current.length());
    var your_product = request.getParameterValues(current)[0];
    var true_product = multiplier * multiplicand;

    out.println("<tr>");
    out.println("<td>" + multiplier + " x " + multiplicand + " = </td>");
    out.println("<td>" + true_product + "</td>");

    if (your_product == true_product) {
        total_score++;
        out.print("<td bgcolor=\"\#00ff00\">");
    }
    else {
        out.print("<td bgcolor=\"\#ff0000\">");
    }
    out.println(your_product + "</td>");
    out.println("</tr>");
}

// This is the main body of the scoring application. It parses the posted quiz,
// and calls the score() function to score remaining problems.

var equations = request.getParameterNames();
while(equations.hasMoreElements()) {
    var currElt = equations.nextElement();
    var splitPos1 = currElt.indexOf("|");
    var splitPos2 = currElt.indexOf(":");

    if (splitPos1 >=0 && splitPos2 >= 0) score(currElt, splitPos1, splitPos2);
}

%>
</table>

<h2>Total Score: <%= total_score %></h2>
<h3><a href="/multiplication_test.jsp">Try again?</a></h3>
</div>

</body>
</html>

```

Web modules

A Web module represents a Web application. A Web module is created by assembling servlets, JavaServer Pages (JSP) files, and static content such as HyperText Markup Language (HTML) pages into a single deployable unit. Web modules are stored in Web archive (WAR) files, which are standard Java archive files.

A Web module contains:

- One or more servlets, JSP files, and HTML files.
- A deployment descriptor, stored in an Extensible Markup Language (XML) file. The file, named `web.xml`, declares the contents of the module. It contains information about the structure and external dependencies of Web components in the module and describes how the components are used at run time.

You can create Web modules as stand-alone applications, or you can combine Web modules with other modules to create J2EE applications. You install and run a Web module in the Web container of an application server.

Assembling Web Modules

If you want to use existing Java 2 Enterprise Edition (J2EE) 1.2 Web modules in your J2EE 1.3 application, migrate them to J2EE 1.3 first.

Assemble a Web module to contain servlets, JSP files, and related code artifacts. (Group enterprise beans, client code, and resource adapter code in separate modules).

A Web module can be installed as a stand-alone application or can be combined with other modules into an enterprise application.

The Application Assembly Tool (AAT) provides flexibility in assembling Web modules. Options described below include:

- Importing an existing Web module (WAR file)
- Creating a new Web module
- Copying code artifacts (such as servlets) from one Web module into a new Web module

Although you can input various properties for Web archives, available properties are specific to the Servlet, JSP, and J2EE specification level.

1. Start the AAT.
2. From the New tab, select **Web Module**. Click **OK**. The navigation tree now displays various sets of properties for configuring the new Web module.
3. Use the property dialog shown in the AAT workspace to change the default file name and location.
 - a. It is recommended that you change the display name so that it differs from the file name.
 - b. If you like, change the temporary location of the Web module from the default location, `install_root/bin`.
4. Add at least one Web component (servlet or JSP file) to the module. You must add at least one Web component, using one of the following methods.
 - Import an existing WAR file containing Web components.
 - a. In the navigation tree, right-click the **Web Components** folder.
 - b. Select **Import** from its right-click menu.

- c. Use the file browser to locate and select the archive file for the module.
 - d. Click **Open**. The Web applications in the selected archive are displayed.
 - e. Select a Web application. Its Web components are displayed in the workspace.
 - f. Select the servlets or JSP files to be added and click **Add**. The components are displayed in the **Selected Components** window.
 - g. Click **OK**. The properties associated with the archive are also imported. The property dialog boxes in the workspace are populated automatically with values.
 - h. Double-click the Web Components icon to verify that the servlets or JSP files are included in the module.
 - i. Double-click the **Web Components** icon to verify that the servlets or JSP files are included in the module.
 - j. Save the Web module.
- Copy and paste archive files from an existing module into the new Web component.
 - Create a new Web component.
 - a. In the navigation tree, right-click the **Web Components** folder.
 - b. Select **New** from its right-click menu.
 - c. When the new module is displayed, enter a component name and choose a component type.
 - d. Use the file browser to locate and select the archive file for the module.
 - e. Click the plus sign (+) to verify its contents and enter assembly properties.
 - f. In the **New Web Component** property dialog box, click **OK**.
 - g. Verify that the Web component has been added to the module by double-clicking the **Web components** icon in the navigation tree.
 - h. Click the component to view its corresponding property dialog box in the bottom portion of the pane.
5. Enter assembly properties for each Web component.
 - a. Click the plus sign (+) next to the each component to reveal its property groups.
 - b. Right-click each property group icon and click **New** to display properties in the workspace.
 6. Specify additional properties for the Web module.

Right-click each property group's icon. Choose **New** to add new values, or edit existing values in the property pane. (Click **Help** for descriptions of the settings).

Note that if you add a security constraint, you must add at least one Web resource collection.
 7. Add any other files needed by the application.
 - In the navigation tree, click the plus sign (+) next to the Files icon. Right-click **Add Class Files**, **Add JAR Files**, or **Add Resource Files**. Select **Add Files**.
 - Add files, using the Add Files dialog.
 8. Save the application.

Assemble other new modules of your choice, if needed:

- Assembling EJB modules.
- Assembling application client modules.
- Assembling resource adapter modules.

You can also migrate existing modules.

Another option is to proceed directly to assembling a new application module. While assembling an application module, you can create any new modules that you need.

Context parameters

A servlet context defines a server's view of the Web application within which the servlet is running. The context also allows a servlet to access resources available to it.

Using the context, a servlet can log events, obtain URL references to resources, and set and store attributes that other servlets in the context can use. These properties declare a Web application's parameters for its context. They convey setup information, such as a webmaster's e-mail address or the name of a system that holds critical data.

Security constraints

Security constraints determine how Web content is to be protected.

These properties associate security constraints with one or more Web resource collections. A constraint consists of a Web resource collection, an authorization constraint and a user data constraint.

- A Web resource collection is a set of resources (URL patterns) and HTTP methods on those resources. All requests that contain a request path that matches the URL pattern described in the Web resource collection is subject to the constraint. If no HTTP methods are specified, then the security constraint applies to all HTTP methods.
- An authorization constraint is a set of roles that users must be granted in order to access the resources described by the Web resource collection. If a user who requests access to a specified URI is not granted at least one of the roles specified in the authorization constraint, the user is denied access to that resource.
- A user data constraint indicates that the transport layer of the client or server communications process must satisfy the requirement of either guaranteeing content integrity (preventing tampering in transit) or guaranteeing confidentiality (preventing reading while in transit).

Servlet mappings

A servlet mapping is a correspondence between a client request and a servlet.

Servlet containers use URL paths to map client requests to servlets, and follow the URL path-mapping rules as specified in the Java Servlet specification. The container uses the URI from the request, minus the context path, as the path to map to a servlet. The container chooses the longest matching available context path from the list of Web applications that it hosts.

Invoker attributes

Invoker attributes are used by the servlet that implements the invocation behavior.

Error pages

Error page locations allow a servlet to find and serve a URI to a client based on a specified error status code or exception type.

These properties are used if the error handler is another servlet or JSP file. The properties specify a mapping between an error code or exception type and the path of a resource in the Web application. The container examines the list in the order that it is defined, and attempts to match the error condition by status code or

by exception class. On the first successful match of the error condition, the container serves back the resource defined in the Location property.

File serving

File serving allows a Web application to serve static file types, such as HTML. File-serving attributes are used by the servlet that implements file-serving behavior.

Initialization parameters

Initialization parameters are sent to a servlet in its `HttpConfig` object when the servlet is first started.

Servlet caching

Dynamic caching can be used to improve the performance of servlet and JavaServer Pages (JSP) files by serving requests from an in-memory cache. Cache entries contain the servlet's output, results of the servlet's execution, and metadata.

Web components

A web component is a servlet, Java Server Page (JSP), or HTML file. One or more web components make up a web module.

Web property extensions

Web property extensions are IBM extensions to the standard deployment descriptors for Web applications. These extensions include mime filtering and servlet caching.

Web resource collections

A Web resource collection defines a set of URL patterns (resources) and HTTP methods belonging to the resource.

HTTP methods handle HTTP-based requests, such as GET, POST, PUT, and DELETE. A URL pattern is a partial Uniform Resource Locator that acts as a template for matching the pattern with existing full URLs in an attempt to find a valid file.

Welcome files

A Welcome file is an entry point file (for example, `index.html`) for a group of related HTML files.

Welcome files are located by using a group of partial URIs. The Web container uses the partial URIs to find a valid file when the initial URI is not found.

Context parameter assembly settings

A servlet context defines the server view of the Web application within which the servlet is running. The context also allows a servlet to access resources available to it. Using the context, a servlet can log events, obtain URL references to resources, and set and store attributes that other servlets in the context can use.

Access this page by traversing the following path in the Application Assembly Tool:

Web Modules > *component_instance*

Parameter name (Required, String)

Specifies the name of a parameter, for example, `dataSourceName`.

Data type String

Parameter value (Required, String)

Specifies the value of a parameter, for example, jdbc/sample.

Data type String

Description

Contains a description of the context parameter.

Data type String

Initialization parameter assembly settings

Use this page to specify the initialization parameters that are sent to a servlet in its `HttpConfig` object when the servlet is first started.

Access this page by traversing the following path in the Application Assembly

Tool: **Web Modules** > *component_instance* > **Web Components** > *component_instance* > **Initialization Parameters**.

Parameter name (Required, String)

Specifies the name of an initialization parameter.

Data type String

Parameter value (Required, String)

Specifies the value of the initialization parameter.

Data type String

Description

Contains text describing the use of the parameter.

Data type String

Filter assembly settings

Use the Filter panel to configure your filter settings.

Access this page by traversing the following path in the Application Assembly

Tool:

Web Modules > *component_instance*

Filter name

Specifies the logical name of the filter. This name maps the filter.

Data type String

Class

Specifies the fully qualified classname of the filter.

Data type String

Description

Provides a description of the filter.

Data type String

JavaServer Pages attribute assembly settings

Use the JavaServer Pages (JSP) attributes page to set JSP attributes that are used by servlets that implement JSP processing behavior.

Access this page by traversing the following path in the Application Assembly Tool:

Web Modules > *component_instance* > **Assembly Property Extensions**

JSP Attribute (Name)

Specifies the name of an attribute.

Data type String

JSP Attribute (Value)

Specifies the value of an attribute.

Data type String

The WebSphere JSP container supports the following JSP attributes:

classdebuginfo

Indicates the compiler should include debugging information in the generated classfile.

classdebuginfo true or false

Default is false.

classpath

Specifies an additional classpath for compiling the generated servlets.

classpath *classpath* or *null*

Default is *null*.

deprecation

Indicates the compiler should generate deprecation warnings when compiling the generated Java source.

deprecation true or false

Default is false.

disableJspRuntimeCompilation

Indicates the runtime behavior of the JSP compiler. If this option is set to true, the JSP compiler does not compile or translate the JSP files, and the JSP engine only loads precompiled classfiles.

disableJspRuntimeCompilation true or false

Default is false.

ieClassID

Indicates the Java plugin COM class ID for Internet Explorer. The <jsp:plugin> tags use this value.

ieClassID *classid*

Default is clsid:8AD9C840-044E-11D1-B3E9-00805F499D93.

javaEncoding

Indicates the Java platform encoding to use to generate the JSP page servlet.

javaEncoding *encoding value*

Default is UTF-8.

jspCompilerPath

Indicates the path of the compiler to use for compiling JSP pages.

jspCompilerPath *path name or null*

Default is *null*.

keepgenerated

Indicates the Java files generated by the JSP compiler during the translation phase of the processing should be kept.

keepgenerated true or false

Default is false.

largefile

Specifies support for large files. When the Java code is generated, the HTML data in a JSP file is stored separately instead of being saved as constant string data in the generated servlet.

largefile true or false

Default is false.

mappedfile

Indicates the compiler should generate Java source that includes a print statement for every line in the JSP file. Use this option for debugging purposes only. It is not recommended for production environments because the mappedfile option generates too many out.print() statements.

mappedfile true or false

Default is false.

scratchdir

Specifies the directory where the generated classfiles are created.

scratchdir *directory name.*

Default is [WAS_INSTALL_ROOT]/temp.

Note: The system property `com.ibm.websphere.servlet.temp.dir` can be used to set the `scratchdir` option on a server-wide basis. This setting, if it is present, overrides the system property.

usePageTagPool

Enables or disables the reuse of custom tag handlers on an individual JavaServer Page basis.

usePageTagPool true or false

Default is false.

The note in the `useThreadTagPool` attribute description also applies to the `usePageTagPool` attribute.

useThreadTagPool

Enables or disables the reuse of custom tag handlers on a per request thread basis.

useThreadTagPool true or false

Default is false.

The note in the `useThreadTagPool` attribute description also applies to the `usePageTagPool` attribute.

Note: Enabling custom tag handler reuse might reveal problems in your tag handler code regarding the tags ability to be reused. A custom tag handler should always do two things:

1. The `release()` method of the tag handler should reset its state and release any private resources that it might have used. The JSP engine guarantees the `release()` method will be called before the tag handler is garbage collected.
2. In the `doEndTag()` method, all instance states associated with this instance must be reset.

verbose

Indicates the compiler should generate verbose output when compiling the generated Java source code.

verbose true or false

Default is false.

Multipurpose Internet Mail Extensions (MIME) filter assembly settings

Use this page to configure Multipurpose Internet Mail Extensions (MIME) filters. Access this page by traversing the following path in the Application Assembly Tool:

Web Modules > *component_instance* > **Assembly Property Extensions**

Component name (Required, String)

Specifies the name of the servlet or JavaServer Pages(TM) (JSP) file. This name must be unique within the Web module.

Data type String

Display name

Specifies a short name that is intended for display by GUIs.

Data type String

Description

Contains a description of the servlet or JSP file.

Data type String

Component type

Specifies the type of Web component. Valid values are servlet or JSP file.

Data type String

Class name (Required, String)

Specifies the full path name for the servlet class.

Data type String

JSP file (Required, String)

Specifies the full path name for the JSP file.

Data type String

Load on startup

Indicates whether this servlet loads at the startup of the Web application. The default is false (the check box is not selected). Also specifies a positive integer indicating the order in which to load the servlet. Lower integers are loaded before higher integers. If no value is specified, or if the value specified is not a positive integer, the container is free to load the servlet at any time in the startup sequence.

Data type String

Small icon

Specifies a JPEG or GIF file containing a small image (16x16 pixels). Use the image as an icon to represent the Web component in a GUI.

Data type JPEG, GIF

Large icon

Specifies a JPEG or GIF file containing a large image (32x32 pixels). Use the image as an icon to represent the Web component in a GUI.

Data type JPEG, GIF

Page list assembly settings

Page lists allow you to avoid hardcoding URLs in servlets and JSP files.

Access this page by traversing the following path in the Application Assembly Tool:

Web Modules > *component_instance* > **Web Components**

Name

Specifies the name of the markup language--for example, Hypertext Markup Language (HTML), Wireless Markup Language (WML), and Voice Extensible Markup Language (VXML).

Data type String

MIME Type

Specifies the Multi-Purpose Internet Mail Extensions (MIME) type of the markup language, for example, text/html and text/x-vxml.

Data type String

Error Page

Specifies the name of an error page.

Data type String

Default Page

Specifies the name of a default page.

Data type String

Pages - Name

Specifies the name of the page to serve, for example, *StockQuoteRequest.page*.

Data type String

Pages - URI

Specifies the URI of the page to serve, for example, *examples/StockQuoteHTMLRequest.jsp*.

Data type String

Security constraint assembly settings

Use the Security constraints panel to configure security constraints.

To view this Application Assembly Tool (AAT) panel, open an existing or create a new Web module. Right-click **Security Constraints** from the left navigation menu. Click **New**.

If multiple security constraints are specified, the container uses the "first match wins" rule when processing a request to determine what authentication method to use, or what authorization to allow

Security constraint name

Specifies the name of the security constraint.

Data type String

Authorization Constraints - Roles

Specifies the user roles that are permitted access to this resource collection.

Data type String

Authorization Constraints - Description

Contains a description of the authorization constraints

Data type String

User Data Constraints - Transport guarantee

Indicates how data communicated between the client and the server is to be protected.

Specifies that the protection for communications between the client and server is None, Integral, or Confidential.

- None means that the application does not require any transport guarantees.
- Integral means that the application requires that the data sent between the client and the server must be sent in such a way that it cannot be changed in transit.
- Confidential means that the application requires that the data must be transmitted in a way that prevents other entities from observing the contents of the transmission.

In most cases, Integral or Confidential indicates that the use of SSL is required.

Data type String

User Data Constraints - Description

Contains a description of the user data constraints.

Data type String

Servlet mapping assembly settings

A servlet mapping is a correspondence between a client request and a servlet. Access this page by traversing the following path in the Application Assembly Tool:

Web Modules > *component_instance*

URL pattern (Required, String)

Specifies the URL pattern of the mapping.

The URL pattern must conform to the Servlet specification. Use the following syntax:

- A string beginning with a slash character (/) and ending with the slash and asterisk characters (/*) represents a path mapping.
- A string beginning with the characters *. represents an extension mapping.
- All other strings are used as exact matches only.
- A string containing only the slash character (/) indicates that the servlet specified by the mapping becomes the default servlet of the application. In this case, the servlet path is the request Uniform Resource Identifier (URI) minus the context path, and the path information is null.

Data type String

Servlet (Required, String)

Specifies the name of the servlet associated with the URL pattern.

Data type String

Tag library assembly settings

Use this page to define the tag library parameters.

Access this page by traversing the following path in the Application Assembly Tool:

Web Modules > *component_instance*

Java ServerPages (JSP) tag libraries contain classes for common tasks such as processing forms and accessing databases from JSP files.

Tag library file name (Required, String)

Specifies a file name relative to the location of the web.xml document, identifying a tag library used in the Web application.

Data type String

Tag library location (Required, String)

Contains the location, as a resource relative to the root of the Web application, where you can find the Tag Library Definition file for the tag library.

Data type String

Welcome file assembly settings

Use this page to configure your welcome page.

Access this page by traversing the following path in the Application Assembly Tool:

Web Modules > *component_instance*

Welcome file (Required, String)

The Welcome file list is an ordered list of partial URLs with no trailing or leading slash characters (/).

The Web server appends each file in the order specified and checks whether a resource in the Web archive (WAR) file is mapped to that request Uniform Resource Identifier (URI). The container forwards the request to the first resource in the WAR file that matches.

Data type String

Servlet caching configuration assembly settings

Use this page to configure your cache groups. Access this page by traversing the following path in the Application AssemblyTool: **Web Modules** > *component_instance* > **Assembly Property Extensions**

The properties on the General tab define a cache group and govern how long an entry remains in the cache. The properties on the ID Generation tab define how

cache IDs are built and the criteria used to cache or invalidate entries. The properties on the Advanced tab define external cache groups and specify custom interfaces for handling servlet caching.

Caching group name

Specifies a name for the group of servlets or JavaServer Pages (JSP) files to cache.

Priority

Defines the default priority for cached servlets. Specify as an integer. The default value is 1.

Priority is an extension of the Least Recently Used (LRU) caching algorithm. It represents the number of cycles through the LRU algorithm that an entry is guaranteed to stay in the cache. The priority represents the length of time that an entry remains in the cache before becoming eligible for removal. On each cycle of the algorithm, the priority of an entry is decremented. When the priority reaches zero, the entry is eligible for invalidation. If an entry is requested while in the cache, its priority is reset to the priority value. Regardless of the priority value and the number of requests, an entry is invalidated when its timeout occurs. Consider increasing the priority of a servlet or JSP file when it is difficult to calculate the output of the servlet or JSP file or when the servlet or JSP file is executed more often than average. Priority values should be low. Higher values do not yield much improvement but use extra LRU cycles. Use timeout to guarantee the validity of an entry. Use priority to rank the relative importance of one entry to other entries. Giving all entries equal priority results in a standard LRU cache that increases performance significantly.

Timeout

Specifies the length of time, in seconds, that a created cache entry remains in the cache.

When this time elapses, the entry is removed from the cache. If the timeout is zero or a negative number, the entry does not time out. It is removed when the cache is full or programmatically, from within an application.

Invalidate only

Specifies that invalidations for a servlet take place, but that no servlet caching is performed.

For example, you can use this property to prevent caching of control servlets. Control servlets treat HTTP requests as commands and execute those commands. By default, this check box is not selected.

Caching group members

Specifies the names of the servlets or JSP files to cache. The URIs are determined from the servlet mappings.

Use URIs for cache ID building

Specifies whether or not to use the URI of the requested servlet to create a cache ID. By default, URIs are used.

Use specified string

Specifies a string representing a combination of request and session variables to use for creating cache IDs. This property defines request and session variables, and the cache uses the values of these variables to create IDs for the entries.

Variables - ID

Specifies the name of a request parameter, request attribute, session parameter, or cookie.

Variables - Type

Specifies the type of variable indicated in the ID field. The valid values are Request parameter, Request attribute, Session parameter, or Cookie.

Variables - Method

Specifies the name of a method in the request attribute or session parameter. The output of this method is used to generate cache entry IDs. If this value is not specified, the toString method is used by default.

Variables - Data ID

Specifies a string that, combined with the value of the variable, generates a group name for the cache entry. The cache entry is placed in this group. You can invalidate this group.

Variables - Invalidate ID

Specifies a string that is combined with the value of the variable on the request or session to form a group name. The cache invalidates the group name.

Required

Specifies whether a value must exist in the request. If this check box is selected, and either the request parameter, request attribute, session parameter, or the method is not specified, the request is not cached.

External cache groups - Group name

Specifies the name of the external cache group to which this servlet is published.

ID generator

Specifies a user-written interface for handling parameters, attributes, and sessions. The value must represent a full package and class name of a class extending com.ibm.websphere.servlet.cache.IdGenerator. The properties specified in the Application Assembly Tool are used and passed to the IdGenerator in the initialize method inside a com.ibm.websphere.servlet.cache.CacheConfig object.

Data type

String

Meta data generator

Specifies a user-written interface for handling invalidation, priority levels, and external cache groups.

The value must represent the full package and class name of a class extending com.ibm.websphere.servlet.cache.MetaDataGenerator. The properties specified in the Application Assembly Tool are used and passed to the MetaDataGenerator in the initialize method inside a com.ibm.websphere.servlet.cache.CacheConfig object.

Data type

String

Web components assembly settings

Use this page to set the assembly properties for the components that make up a Web module.

Access this page by traversing the following path in the Application Assembly Tool:

Web Modules > *component_instance* > **Web Components**

Component name

Specifies the name of the servlet or JavaServer Pages(TM) (JSP) file. This name must be unique within the Web module.

Data type String

Display name

Specifies a short name that is intended for display by GUIs.

Data type String

Description

Contains a description of the servlet or JSP file.

Data type String

Component type

Specifies the type of Web component. Valid values are servlet or JSP file.

Data type String

Class name

Specifies the full path name for the servlet class.

Data type String

JSP file

Specifies the full path name for the JSP file.

Data type String

Load on startup

Indicates whether this servlet loads at the startup of the Web application. The default is false (the check box is not selected).

This field also specifies a positive integer indicating the order in which the servlet is to load. Lower integers are loaded before higher integers. If no value is specified, or if the value specified is not a positive integer, the container is free to load the servlet at any time in the startup sequence.

Data type Boolean

Default False

Small icon

Specifies a JPEG or GIF file containing a small image (16x16 pixels). Use the image as an icon to represent the Web component in a GUI.

Data type String

Large icon

Specifies a JPEG or GIF file containing a large image (32x32 pixels). Use the image as an icon to represent the Web component in a GUI.

Data type String

Run as role name

Enter a role name that represents the user account under which the servlet executes. The default role name is blank, which indicates the servlet runs under the user that logged into the application server. The role name of "all role" indicates the servlet can execute under different users.

Data type String

Description

In this optional field, enter a description that explains the importance of the role, and where and how the role can be used.

Data type String

Run as role mode

Indicates a security role that is defined in the enterprise application.

Data type String

Local Transactions - Unresolved action

Specifies the action the Web container must take if resources in a local transaction are uncommitted by an application. This property is an IBM extension to the standard J2EE deployment descriptor. A local transaction context is provided by the container in the absence of a global transaction context.

Data type	String
Default	Rollback
Range valid values are	Commit Rollback

Additional information about these settings follows:

Commit

At end of the local transaction context, the container instructs all the unresolved local transactions to commit.

Rollback (default)

At end of the local transaction context, the container instructs all the unresolved local transactions to rollback.

Web modules assembly settings

Use this page to set the assembly properties for web modules. Web modules are composed of one or more web components.

Access this page by traversing the following path in the Application Assembly Tool:

application_instance > **Web Modules**

File name

Specifies the file name of the Web module, relative to the top level of the application package.

Alternative DD

Specifies the file name for an alternative deployment descriptor file to use instead of the original deployment descriptor file in the module's JAR file.

This file is the post-assembly version of the deployment descriptor file. (The original deployment descriptor file can be edited to resolve dependencies and security information. Directing the use of the alternative deployment descriptor

allows you to keep the original deployment descriptor file intact). The value of the Alternative DD property must be the full path name of the deployment descriptor file relative to the module's root directory. By convention, the file is in the ALT-INF directory. If this property is not specified, the deployment descriptor file is read directly from the module's JAR file.

Context root

Specifies the context root of the Web application. The context root is combined with the defined servlet mapping (from the WAR file) to compose the full URL that users type to access the servlet.

For example, if the context root is /gettingstarted and the servlet mapping is MySession, then the URL is http://host:port/gettingstarted/MySession.

Classpath

Specifies the class path for resources used by the Web application, relative to the ear file..

If your Web application requires access to classes within an ear file, specify the relative path of the classes in this field.

Display name

Specifies a short name that is intended to be displayed by GUIs.

Description

Contains a description of the Web module.

Distributable

Specifies that this Web application is programmed appropriately to deploy into a distributed servlet container.

Small icon

Specifies a JPEG or GIF file containing a small image (16x16 pixels). The image is used as an icon to represent the module in a GUI.

Large icon

Specifies a JPEG or GIF file containing a large image (32x32 pixels). The image is used as an icon to represent the module in a GUI.

Session configuration

Indicates that session configuration information is present. Checking this box makes the Session timeout property editable.

Session timeout

Specifies a time period, in seconds, after which a client is considered inactive. The default value is zero, indicating that the session timeout never expires.

Login configuration -- Authentication method

Specifies an authentication method to use. As a prerequisite to gaining access to any Web resources protected by an authorization constraint, a user must authenticate by using the configured mechanism.

A Web application can authenticate a user to a Web server by using one of the following mechanisms: HTTP basic authentication, HTTP digest authentication, HTTPS client authentication, and form-based authentication.

- HTTP basic authentication is not a secure protocol because the user password is transmitted with a simple Base64 encoding and the target server is not authenticated. In basic authentication, the Web server requests a Web client to authenticate the user and passes a string called the realm of the request in which the user is to be authenticated.
- HTTP digest authentication transmits the password in encrypted form.

- HTTPS client authentication uses HTTPS (HTTP over SSL) and requires the user to possess a public key certificate.
- Form-based authentication allows the developer to control the appearance of login screens.

The Login configuration properties are used to configure the authentication method that should be used, the realm name that should be used for HTTP basic authentication, and the attributes that are needed by the form-based login mechanism. Valid values for this property are Unspecified, Basic, Digest, Form, and Client certification.

Note:

HTTP digest authentication is not supported as a login configuration in this product. Also, not all login configurations are supported in all of the product's global security authentication mechanisms (Local Operating system, LTPA, and custom pluggable user registry). HTTP basic authentication and form-based login authentication are the only authentication methods supported by the Local Operating system user registry. LTPA and the custom pluggable user registry are capable of supporting HTTP basic authentication, form-based login, and HTTPS client authentication.

Login configuration -- Realm name

Specifies the realm name to use in HTTP basic authorization. It is based on a user name and password, sent as a string (with a simple Base64 encoding). An HTTP realm is a string that allows URIs to be grouped together. For example, if a user accesses a secured resource on a Web server within the "finance realm," subsequent access to the same or different resource within the same realm does not result in a repeat prompt for a user ID and password.

Login configuration -- Login page

Specifies the location of the login form. If form-based authentication is not used, this property is disabled.

Form Login Config -- Error page

Specifies the location of the error page. If form-based authentication is not used, this property is disabled.

Reload interval

Specifies a time interval, in seconds, in which the file system of the Web application is scanned for updated files. The default is 3 seconds.

Reloading enabled

Specifies whether file reloading is enabled. The default is true.

Default error page

Specifies a file name for the default error page. If no other error page is specified in the application, this error page is used.

Additional classpath

Specifies the full class path that will be used to reference classes outside of those specified in the archive.

If your Web application requires access to classes not contained in the archive file, specify the full path for those classes in this field.

File serving enabled

Specifies whether file serving is enabled. File serving allows the application to serve static file types, such as HTML and GIF. File serving can be disabled if the application contains only dynamic components. The default value is true.

Directory browsing enabled

Specifies whether directory browsing is enabled. Directory browsing allows the application to browse disk directories. Directory browsing can be disabled if, you want to protect data. The default value is true.

Serve servlets by classname

Specifies whether a servlet can be served by requesting its class name. Usually, servlets are served only through a URI reference. The class name is the actual name of the servlet on disk. For example, a file named SnoopServlet.java compiles into SnoopServlet.class. (This is the class name.) SnoopServlet.class is normally invoked by specifying snoop in the URL. However, if the *Serve servlets by classname* property is enabled, the servlet is invoked by specifying SnoopServlet. The default value is true.

Virtual hostname

Specifies a virtual host name. A virtual host is a configuration enabling a single host machine to resemble multiple host machines. This property allows you to bind the application to a virtual host in order to enable execution on that virtual host.

Filter mappings

Specifies the filter mapping declarations in this application. The container uses the filter mapping declarations to decide on the type and order of filters to apply to a request.

After the container matches the request URI to a servlet, for each filter mapping element, it determines what filters to apply based on the servlet name or the URL pattern, depending on the style specified. Filters are invoked in the same order as the one specified in the list of filter mapping elements. The value that you specify for the filter name must be the same value as that specified in the `<filter><filename>` sub-element declarations in the deployment descriptor.

Assembly property extensions

Use this panel to configure WebSphere Application Server specific Web module extensions, or also referred to as assembly property extensions.

This panel lists the extensions that can be configured through the tool.

Reach the applicable extension panel by clicking on the panel name in the navigation at the left, or by double-clicking the attribute name in the list provided.

Access this page by traversing the following path in the Application AssemblyTool:

Web Modules > *component_instance* > **Assembly Property Extensions**

Assembly properties for a Web module include:

- File serving attributes
- Invoker attributes
- JavaServer Pages (JSP) attributes
- Multipurpose Internet Mail Extensions (MIME) filters
- Servlet caching configurations

File serving attribute assembly settings

File serving allows a Web application to serve static file types, such as HTML. File-serving attributes are used by the servlet that implements file-serving behavior. Access this page by traversing the following path in the Application AssemblyTool:

Web Modules > *component_instance* > **Assembly Property Extensions**

File Serving Attribute (Name)

Specifies the name of an attribute.

Data type String

File Serving Attribute (Value)

Specifies the value of an attribute.

Data type String

Invoker attribute assembly settings

Invoker attributes are used by the servlet that implements the invocation behavior. Access this page by traversing the following path in the Application Assembly Tool:

Web Modules > *component_instance* > **Assembly Property Extensions**

Invoker Attribute (Name)

Specifies the name of an attribute.

Data type String

Invoker Attribute (Value)

Specifies the value of an attribute.

Data type String

Error page assembly settings

Error page locations allow a servlet to find and serve a URI to a client based on a specified error status code or exception type. These properties are used if the error handler is another servlet or JSP file.

Access this page by traversing the following path in the Application Assembly Tool:

Web Modules > *component_instance*

The error page properties specify a mapping between an error code or exception type and the path of a resource in the Web application. The container examines the list in the order that it is defined, and attempts to match the error condition by status code or by exception class. On the first successful match of the error condition, the container serves back the resource defined in the Location property.

Error code

Indicates that the error condition is a status code.

Data type Integer

Error Code (Required, String)

Specifies an HTTP error code, for example, 404.

Data type String

Exception

Indicates that the error condition is an exception type.

Data type String

Exception type name (Required, String)

Specifies an exception type.

Data type String

Location (Required, String)

Contains the location of the error-handling resource in the Web application.

Data type String

Web resource collections security constraint properties

A Web resource collection defines a set of URL patterns or resources and HTTP methods belonging to the resource, which define the security constraints for a Web component.

Access this page by traversing the following path in the Application Assembly Tool:

Web Modules > *component_instance*

HTTP methods handle HTTP-based requests, such as GET, POST, PUT, and DELETE. A URL pattern is a partial Uniform Resource Locator that acts as a template for matching the pattern with existing full URLs in an attempt to find a valid file.

Web resource name

Specifies the name of a Web resource collection.

Data type String

Web resource description

Contains a description of the Web resource collection.

HTTP methods

Specifies the HTTP methods to which the security constraints apply. If no HTTP methods are specified, then the security constraint applies to all HTTP methods. The valid values are GET, POST, PUT, DELETE, HEAD, OPTIONS, and TRACE.

Data type String

URL pattern

Specifies URL patterns for resources in a Web application. All requests that contain a request path that matches the URL pattern are subject to the security constraint.

Data type String

Troubleshooting tips for Web application deployment

Deployment of a Web application is successful if you can access the application by typing a Uniform Resource Locator (URL) in a browser, or if you can access the application by following a link.

If you cannot access your application, follow these steps to eliminate some common errors that can occur during migration or deployment.

Web module does not run in WebSphere Application Server Version 5.

Symptom	Your Web module does not run when you migrate it to Version 5
Problem	In Version 4.x, the classpath setting that affected visibility was <i>Module Visibility Mode</i> . In Version 5, you must use class loader policies to set visibility.
Recommended response	Reassemble an existing module, or change the visibility settings in the class loader policies. in the class loader policies. See article Migration of module visibility modes from Version 4.x for more information and examples.

Welcome page is not visible.

Symptom	You cannot access an application with a Web path of: <code>/webapp/myapp</code>
Problem	The default welcome page for a Web application is assumed to be <i>index.html</i> . You cannot access the default page of the <i>myapp</i> application unless it is named <i>index.html</i> .
Recommended response	To identify a different welcome page, modify the properties of the Web module during assembly. See article Assembling Web modules for more information.

HTML files are not found.

Symptom	Your Web application ran successfully on prior versions, but now you encounter errors that the welcome page (typically <i>index.html</i>), or referenced HTML files are not found: Error 404: File not found: Banner.html Error 404: File not found: HomeContent.html
----------------	--

Problem

For security and consistency reasons, Web application URLs are now case-sensitive on all operating systems.

Suppose the content of the index page is as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 5.0 Frameset//EN">
<HTML>
<TITLE>
Insurance Home Page
</TITLE>
<frameset rows="18,80">
  <frame src="Banner.html" name="BannerFrame" SCROLLING=NO>
  <frame src="HomeContent.html" name="HomeContentFrame">
</frameset>
</HTML>
```

However the actual file names in the `\WebSphere\AppServer\installedApps\...` directory where the application is deployed are:

```
banner.html
homecontent.html
```

Recommended response

To correct this problem, modify the `index.html` file to change the names `Banner.html` and `HomeContent.html` to `banner.html` and `homecontent.html` to match the names of the files in the deployed application.

Modifying the default Web container configuration

The Web container is created initially with default properties values suitable for simple Web applications. However, these values might not be appropriate for more complex Web applications.

Your application is considered complex if it requires any of the following features:

- virtual host
- servlet caching
- special client request loads
- persistent HTTP session support
- special HTTP transport settings
- transaction class mappings

Modify the following properties if you have a complex application:

1. If your Web application requires a virtual host, other than the default `_host`, or requires servlet caching, modify the Web container **General Properties**.
2. If your application handles special client request loads, modify the Web Container **Additional Properties** > **Thread Pool** setting.
3. If your application requires persistent HTTP session support, modify the Web Container **Additional Properties** > **Session Management** setting.
4. If your application requires one of the following HTTP transport settings:
 - Unique hostname and port for client access
 - SSL enablement

modify the Web Container **Additional Properties** > **HTTP transports** setting.

5. If your application requires global settings for internal servlets for WAR files packaged by third-party tools, modify the Web Container **Additional Properties** > **Custom Properties** setting.

6. If your application uses transaction class mappings to classify workload, modify the Web Container **Additional Properties > Advanced Settings**.

Web container

A Web container handles requests for servlets, JavaServer Pages (JSP) files, and other types of files that include server-side code. The Web container creates servlet instances, loads and unloads servlets, creates and manages request and response objects, and performs other servlet management tasks.

The Web server plug-ins, provided by the WebSphere Application Server, help supported Web servers pass servlet requests to Web containers.

Web container settings

Use this page to configure the container settings.

Access this page by traversing the following path in the administrative console:

Servers > Application Servers > *server_instance* > Web container

Configuration - General Properties

Default virtual host

Specifies a virtual host is a configuration enabling a single host machine to resemble multiple host machines. Resources associated with one virtual host cannot share data with resources associated with another virtual host, even if the virtual hosts share the same physical machine.

Select a virtual host option:

Default Host

The product provides a default virtual host with some common aliases, such as the machine IP address, short host name, and fully qualified host name. The alias comprises the first part of the path for accessing a resource such as a servlet. For example, it is `localhost:9080` in the request `http://localhost:9080/myServlet`.

Admin Host

This is another name for the application server; also known as *server1* in the base installation. This process supports the use of the administrative console.

Servlet caching

Specifies that if a servlet is invoked once and it generates output to be cached, a cache entry is created containing not only the output, but also side effects of the invocation. These side effects can include calls to other servlets or Java Server Pages (JSP) files, as well as metadata about the entry, including timeout and entry priority information.

Enable servlet caching

Check this box to enable servlet caching.

Web module settings

Use this page to configure Web module settings.

Access this page by traversing the following path in the administrative console:

Applications > Enterprise Application > *application_instance* > Web Module

URI

Specifies a URI that, when resolved relative to the application URL, specifies the location of the module archive contents on a file system. The URI must match the ModuleRef URI in the deployment descriptor of an application if the module was packaged as part of a deployed application or enterprise archive (EAR) file.

Name

Specifies the unique display name for the module.

Alternate DD

Specifies the file name for an alternative deployment descriptor file to use instead of the original deployment descriptor file in the module JAR file.

This file is the *post-assembly* version of the deployment descriptor file. You can edit the original deployment descriptor file to resolve dependencies and security information. Specifying the use of the alternative deployment descriptor keeps the original deployment descriptor file intact.

The value of the *Alternate DD* property must be the full path name of the deployment descriptor file, relative to the module root directory. By convention, the file is in the ALT-INF directory. If this property is not specified, the deployment descriptor file is read from the module JAR file.

Starting weight

Specifies the order in which modules are started. Lower weighted modules are started before higher weighted modules.

Prefer WEB-INF Classes

Specifies classes to load in WEB-INF before any other classes. Implementing the application class loader is recommended so that classes and resources packaged within the WAR file load before classes and resources residing in container-wide library JAR files.

Initial State

Specifies the default state of this application at server startup.

Web Module Deployment settings

Use this page to configure an instance of Web module deployment.

To view this administrative console page, click **Applications > Enterprise Application > application_instance > Web Modules > Web Module_instance**.

URI

Specifies a URI that, when resolved relative to the application URL, specifies the location of the module archive contents on a file system. The URI must match the ModuleRef URI in the deployment descriptor of an application if the module was packaged as part of a deployed application or enterprise archive (EAR) file.

Alternate DD

Specifies the file name for an alternative deployment descriptor file to use instead of the original deployment descriptor file in the module JAR file.

This file is the *post-assembly* version of the deployment descriptor file. You can edit the original deployment descriptor file to resolve dependencies and security information. Specifying the use of the alternative deployment descriptor keeps the original deployment descriptor file intact.

The value of the *Alternate DD* property must be the full path name of the deployment descriptor file, relative to the module root directory. By convention, the file is in the ALT-INF directory. If this property is not specified, the deployment descriptor file is read from the module JAR file.

Starting weight

Specifies the order in which modules are started. Lower weighted modules are started before higher weighted modules.

ClassLoader Mode

Specifies whether the class loader should search in the parent class loader or in the application class loader first to load a class. The standard for JDK class loaders and WebSphere class loaders is PARENT_FIRST. By specifying PARENT_LAST, your application can override classes contained in the parent class loader, but this action can potentially result in ClassCastException or LinkageErrors if you have mixed use of overridden classes and non-overridden classes.

The options are PARENT_FIRST and PARENT_LAST. The default is to search in the parent class loader before searching in the application class loader to load a class.

Data type	String
Default	PARENT_FIRST

Web container advanced settings

Use this page to support Web container advanced settings. This support includes Network QoS and transaction class mapping

To view this administrative console page, click **Servers > Application Servers > *server name* > Web Container > Advanced Settings**.

Network QoS

Specifies the parameter that will be used to classify outbound data that is delivered in response to HTTP and HTTPS requests.

The classification parameters are used to construct an ApplicationData parameter for the TCP/IP network service, which is called Quality of Service (QoS). The ApplicationData parameter is used in a QoS PolicyRule statement.

You can specify at most one classification parameter. If you do not specify a classification parameter, the response data will not be classified to the network agent.

Parameter	Description
HOST	Indicates that the Host value from the Host header, not including the port, is to be used used to construct an ApplicationData parameter. If you specify this parameter, WebSphere for z/OS classifies the outbound response data by using the HOST value. In the request: <code>http://www.mycompany.com/mywebap/myservlet</code> <code>www.mycompany.com</code> represents the host value.

URI	<p>Indicates that the part of the Universal Resource Locator that specifies the path to a resource is to be used to construct an <code>ApplicationData</code> parameter. If you specify this parameter, WebSphere for z/OS classifies the outbound response data by using the URI value. The path must be specified exactly as it is entered in a browser because the check for this path is case sensitive.</p> <p>In the request: <code>http://www.mycompany.com/mywebap/myservlet</code></p> <p><code>/mywebap/myservlet</code> represents the URI value.</p>
HOSTURI	<p>Indicates that the <code>HOST</code> and <code>URI</code>, concatenated together, are to be used to construct an <code>ApplicationData</code> parameter. If you specify this parameter, WebSphere for z/OS classifies the outbound response using the concatenated <code>HOST</code> and <code>URI</code> value.</p> <p>In the request: Get request: <code>http://www.mycompany.com/mywebap/myservlet</code></p> <p><code>www.mycompany.com/mywebap/myservlet</code> represents the concatenated <code>HOST</code> and <code>URI</code> value.</p>
TCLASS	<p>Indicates that a valid Workload Management (WLM) transaction class is to be used to construct an <code>ApplicationData</code> parameter. If you specify this parameter, you must specify the fully qualified name of the transaction class mapping file on the Transaction Class Mapping property.</p>

Transaction Class Mapping

Specifies the fully qualified name of the file that contains the rules for classifying the Workload Management Transaction Class for HTTP or HTTPS requests. The file name is class sensitive.

For example, if `tclass.conf` is the name of your transaction class mapping file, you would specify the following for the value on this property:

```
/mydir/tclass.conf
```

where *mydir* is the fully qualified directory where the `tclass.conf` file is located.

For example

```
/mydir/tclass.conf
```

Web container custom property settings

Use this page to configure arbitrary name-value pairs of data, where the name is a property key and the value is a string value that can be used to set internal system configuration properties. Defining a new property enables you to configure a setting beyond that which is available in the administrative console.

To view this administrative console page, click **Servers > Application Servers > *server_name* > Web Container > Custom Properties**.

Name

Specifies the name (or key) for the property.

Data type String

Value

Specifies the value paired with the specified name.

Data type String

Description

Provides information about the name-value pair.

Data type String

Web Archive (WAR) files packaged using third-party tools cannot specify behavior for the services exposed by the Web container internal servlets. You can globally enable/disable internal servlets for all Web applications at the Web container level by creating name-value pairs such as:

Name	Value
fileServingEnabled	true
directoryBrowsingEnabled	true
serveServletsByClassnameEnabled	true

Settings defined at the Application Assembly Tool level take precedence over the global settings set through the custom properties at the Web container level.

Web application deployment extensions continue to hold configuration information for the services provided by the internal servlets, and take precedence over the global settings set through the custom properties at the Web container level.

Web applications: Resources for learning

Use the following links to find relevant supplemental information about Web applications. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

View links to additional information about:

- "Web applications: Resources for learning"
- "Web applications: Resources for learning"
- "Web applications: Resources for learning"

Programming model and decisions

- J2EE BluePrints for Web applications
- Redbook on the design and implementation of Servlets, JSP files, and enterprise beans

Programming instructions and examples

- Redbook on Servlet and JSP file Programming
- Sun's Java™ Tutorial on Servlets
- Introduction to JavaServer Pages - Tutorial
- Bean Scripting Framework description
- Web delivered samples in the Samples Gallery

Programming specifications

- Java 2 Software Development Kit (SDK)
- Servlet 2.3 Specification
- JavaServer Pages 1.2 Specification
- Differences between JavaScript and ECMAScript
- ISO 8859 Specifications

Chapter 3. Managing HTTP sessions

IBM WebSphere Application Server provides a service for managing HTTP sessions: Session Manager. The key activities for session management are summarized below.

Before you begin these steps, make sure you are familiar with the programming model for accessing HTTP session support in the applications following the Servlet 2.3 API.

1. Plan your approach to session management, which could include session tracking, session recovery, and session clustering.
2. Create or modify your own applications to use session support to maintain sessions on behalf of Web applications.
3. Assemble your application.
4. Deploy your application.
5. Ensure the administrator appropriately configures session management in the administrative domain.
6. Adjust configuration settings and perform other tuning activities for optimal use of sessions in your environment.

Sessions

A session is a series of requests to a servlet, originating from the same user at the same browser.

Sessions allow applications running in a Web container to keep track of individual users.

For example, a servlet might use sessions to provide "shopping carts" to online shoppers. Suppose the servlet is designed to record the items each shopper indicates he or she wants to purchase from the Web site. It is important that the servlet be able to associate incoming requests with particular shoppers. Otherwise, the servlet might mistakenly add Shopper_1's choices to the cart of Shopper_2.

A servlet distinguishes users by their unique session IDs. The session ID arrives with each request. If the user's browser is cookie-enabled, the session ID is stored as a cookie. As an alternative, the session ID can be conveyed to the servlet by URL rewriting, in which the session ID is appended to the URL of the servlet or JavaServer Pages (JSP) file from which the user is making requests. For requests over HTTPS or Secure Sockets Layer (SSL), Another alternative is to use SSL information to identify the session.

Migrating HTTP sessions

Note: In Version 5 default write frequency mode is `TIME_BASED_WRITES`, which is different from Version 4.0 and 3.5 default mode of `END_OF_SERVICE`.

Migrating from Version 4.0

No programmatic changes are required to migrate from version 4.0 to version 5.

Migrating from Version 3.5

If you have Version 3.5 applications running in Servlet 2.1 mode, some of the following Version 5 differences might influence how you choose to track and manage sessions.

1. During application development, modify session-related APIs as needed.

Some API changes are required in order to redeploy existing applications on Version 5. These include changes to the HttpSession API itself as well as issues associated with moving to support for the Servlet 2.3 specification. Certain Servlet 2.1 API methods have been deprecated in Servlet 2.3 API. These deprecated APIs still work in Version 5.0, but they may be removed in a future version of the API. Changes are summarized in the following list:

- Replace instances of `getValue()` with `getAttribute()`
- Replace instances of `getValueNames()` with `getAttributeNames()`
- Replace instances of `removeValue()` with `removeAttribute()`
- Replace instances of `putValue()` with `setAttribute()`

2. During application development, modify Web application behavior as needed.

In accordance with the Servlet 2.3 specification, HttpSession objects must be scoped within a single Web application context; they may not be shared between contexts. This means that a session can no longer span Web applications. Objects added to a session by a servlet or JSP in one Web application cannot be accessed from another Web application. The same session ID may be shared (because the same cookie is in use), but each Web application will have a unique session associated with the session ID. Version 5 provides a feature that can be used to extend scope of a session to enterprise application.

3. Use administrative tools to configure Session Manager security settings as needed. Relative to session security, the default Session Manager setting for Integrate Security is now false. This is different from the default setting in some earlier releases.

4. Use administrative tools to configure the JSP enabler and application server as needed.

In Version 3.5 of the product, JSP files that contained the usebean tag with scope set to session did not always work properly when session persistence was enabled. Specifically, the JSP writer needed to write a scriptlet to explicitly set the attribute (that is, to call `setAttribute()`) if it was changed as part of JSP processing.

Two new features in Version 5.0 help address this problem:

- You can set `dosetattribute` to true on the JSP InitParameter.
- You can set the Write Contents option to Write all.

The differences between the two solutions are summarized in the following table:

	Applies to	Configured at	Action
<code>dosetattribute</code> set to <i>true</i>	JSP	JSP enabler	Assures that JSP session-scoped beans always call <code>setAttribute()</code>
Write Contents option set to <i>Write all</i>	servlet or JSP	application server	All session data (changed or unchanged) is written to the external location

If session persistence is enabled and a class reload for the Web application occurs, the sessions associated with the Web application are maintained in the persistent store and will be available after the reload.

Developing session management in servlets

This information, combined with the coding example `SessionSample.java`, provides a programming model for implementing sessions in your own servlets.

1. Get the `HttpSession` object.

To obtain a session, use the `getSession()` method of the `javax.servlet.http.HttpServletRequest` object in the Java Servlet 2.3 API.

When you first obtain the `HttpSession` object, the Session Management facility uses one of three ways to establish tracking of the session: cookies, URL rewriting, or Secure Sockets Layer (SSL) information.

Assume the Session Management facility uses cookies. In such a case, the Session Management facility creates a unique session ID and typically sends it back to the browser as a *cookie*. Each subsequent request from this user (at the same browser) passes the cookie containing the session ID, and the Session Management facility uses this ID to find the user's existing `HttpSession` object.

In Step 1 of the code sample, the `Boolean(create)` is set to `true` so that the `HttpSession` object is created if it does not already exist. (With the Servlet 2.3 API, the `javax.servlet.http.HttpServletRequest.getSession()` method with no boolean defaults to `true` and creates a session if one does not already exist for this user.)

2. Store and retrieve user-defined data in the session.

After a session is established, you can add and retrieve user-defined data to the session. The `HttpSession` object has methods similar to those in `java.util.Dictionary` for adding, retrieving, and removing arbitrary Java objects.

In Step 2 of the code sample, the servlet reads an integer object from the `HttpSession`, increments it, and writes it back. You can use any name to identify values in the `HttpSession` object. The code sample uses the name `sessiontest.counter`.

Because the `HttpSession` object is shared among servlets that the user might access, consider adopting a site-wide naming convention to avoid conflicts.

3. (Optional) Output an HTML response page containing data from the `HttpSession` object.

4. Provide feedback to the user that an action has taken place during the session.

You may want to pass HTML code to the client browser indicating that an action has occurred. For example, in step 3 of the code sample, the servlet generates a Web page that is returned to the user and displays the value of the `sessiontest.counter` each time the user visits that Web page during the session.

5. (Optional) Notify Listeners. Objects stored in a session that implement the `javax.servlet.http.HttpSessionBindingListener` interface are notified when the session is preparing to end and become invalidated. This notice enables you to perform post-session processing, including permanently saving the data changes made during the session to a database.

6. End the session. You can end a session:

- Automatically with the Session Management facility if a session is inactive for a specified time. The administrators provide a way to specify the amount of time after which to invalidate a session.
- By coding the servlet to call the `invalidate()` method on the session object.

Example: SessionSample.java

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SessionSample extends HttpServlet {
    public void doGet (HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        // Step 1: Get the Session object

        boolean create = true;
        HttpSession session = request.getSession(create);

        // Step 2: Get the session data value

        Integer ival = (Integer)
        session.getAttribute ("sessiontest.counter");
        if (ival == null) ival = new Integer (1);
        else ival = new Integer (ival.intValue () + 1);
        session.setAttribute ("sessiontest.counter", ival);

        // Step 3: Output the page

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>Session Tracking Test</title></head>");
        out.println("<body>");
        out.println("<h1>Session Tracking Test</h1>");
        out.println ("You have hit this page " + ival + " times" + "<br>");
        out.println ("Your " + request.getHeader("Cookie"));
        out.println("</body></html>");
    }
}
```

Assembling so that session data can be shared

In accordance with the Servlet 2.3 API specification, by default the Session Management facility supports session scoping by Web module. Only servlets in the same Web module can access the data associated with a particular session. WebSphere Application Server provides an option that you can use to extend the scope of the session attributes to an enterprise application. Therefore, you can share session attributes across all the Web modules in an enterprise application. This option is provided as an IBM extension.

Restriction: To use this option, you must install all the Web modules in the enterprise application on a given server. You cannot split up Web modules in the enterprise application by servers. For example, with an enterprise application containing two Web modules, you cannot use this option when one Web module is installed on one server and second Web module is installed on a different server. In such split installations, applications might share session attributes across Web modules using distributed sessions, but session data integrity is lost when concurrent access to a session is made in different Web modules. It also severely restricts use of some Session Management features, like TIME_BASED_WRITES. For enterprise applications on which this option is enabled, the Session Management configuration on the Web module inside the enterprise application is

ignored. Then Session Management configuration defined on enterprise application is used if Session Management is overwritten at the enterprise application level. Otherwise, the Session Management configuration on the Web container is used.

Servlet API Behavior

Note: If shared HttpSession context is turned on in an enterprise application, HttpSession listeners defined in all the Web modules inside the enterprise application are invoked for session events. The order of listener invocation is not guaranteed.

Do the following to share session data across Web modules in an enterprise application:

1. Launch the Application Assembly Tool (AAT).
2. Click the application (EAR file) you want to share.
3. Click the **IBM extension tab**.
4. Click **Shared httpsession context**.
5. Click **Apply**. Make sure the class definition of attributes put into session are available to all the Web modules in the enterprise application.
6. Save the application (EAR) file.

Session security support

You can integrate HTTP sessions and security in IBM WebSphere Application Server. When security integration is enabled in the Session Management facility and a session is accessed in a protected resource, you can access that session only in protected resources from then on. You cannot mix secured and unsecured resources accessing sessions when security integration is turned on. Security integration in the Session Management facility is not supported in form-based login with SWAM.

Security integration rules for HTTP sessions

Only authenticated users can access sessions created in secured pages and are created under the identity of the authenticated user. Only this authenticated user can access these sessions in other secured pages. To protect these sessions from unauthorized users, you cannot access them from an unsecure page.

Programmatic details and scenarios

IBM WebSphere Application Server maintains the security of individual sessions.

An identity or user name, readable by the `com.ibm.websphere.servlet.session.IBMSession` interface, is associated with a session. An unauthenticated identity is denoted by the user name `anonymous`. IBM WebSphere Application Server includes the `com.ibm.websphere.servlet.session.UnauthorizedSessionRequestException` class, which is used when a session is requested without the necessary credentials.

The Session Management facility uses the WebSphere Application Server security infrastructure to determine the authenticated identity associated with a client HTTP request that either retrieves or creates a session. WebSphere Application Server security determines identity using certificates, LPTA, and other methods.

After obtaining the identity of the current request, the Session Management facility determines whether to return the session requested using a `getSession()` call or not.

The following table lists possible scenarios in which security integration is enabled with outcomes dependent on whether the HTTP request is authenticated and whether a valid session ID and user name was passed to the Session Management facility.

* A `com.ibm.websphere.servlet.session.UnauthorizedSessionRequestException` error is thrown to the servlet.

	Unauthenticated HTTP request is used to retrieve a session	HTTP request is authenticated, with an identity of "FRED" used to retrieve a session
No session ID was passed in for this request, or the ID is for a session that is no longer valid	A new session is created. The user name is anonymous	A new session is created. The user name is FRED
A session ID for a valid session is passed in. The current session user name is "anonymous"	The session is returned.	The session is returned. Session Management changes the user name to FRED
A session ID for a valid session is passed in. The current session user name is FRED	The session is not returned. An <code>UnauthorizedSessionRequestException</code> error is thrown*	The session is returned.
A session ID for a valid session is passed in. The current session user name is BOB	The session is not returned. An <code>UnauthorizedSessionRequestException</code> error is thrown*	The session is not returned. An <code>UnauthorizedSessionRequestException</code> error is thrown*

Session management support

WebSphere Application Server provides facilities, grouped under the heading *Session Management*, that support the `javax.servlet.http.HttpSession` interface described in the Servlet API specification.

In accordance with the Servlet 2.3 API specification, the Session Management facility supports session scoping by Web module. Only servlets in the same Web module can access the data associated with a particular session. Multiple requests from the same browser, each specifying a unique Web application, result in multiple sessions with a shared session ID. You can invalidate any of the sessions that share a session ID without affecting the other sessions.

You can configure a session timeout for each Web application. A Web application timeout value of 0 (the default value) means that the invalidation timeout value from the Session Management facility is used.

When an HTTP client interacts with a servlet, the state information associated with a series of client requests is represented as an HTTP session and identified by a session ID. Session Management is responsible for managing HTTP sessions, providing storage for session data, allocating session IDs, and tracking the session ID associated with each client request through the use of cookies or URL rewriting techniques. Session Management can store session-related information in several ways:

- In application server memory (the default). This information cannot be shared with other application servers.
- In a database. This storage option is known as *database persistent sessions*.
- In another WebSphere Application Server instance. This storage option is known as *memory-to-memory sessions*.

The last two options are referred to as *distributed sessions*. Distributed sessions are essential for using HTTP sessions for failover facility. When an application server receives a request associated with a session ID that it currently does not have in memory, it can obtain the required session state by accessing the external store (database or memory-to-memory). If distributed session support is not enabled, an application server cannot access session information for HTTP requests that are sent to servers other than the one where the session was originally created. Session Management implements caching optimizations to minimize the overhead of accessing the external store, especially when consecutive requests are routed to the same application server.

Storing session states in an external store also provides a degree of fault tolerance. If an application server goes offline, the state of its current sessions is still available in the external store. This availability enables other application servers to continue processing subsequent client requests associated with that session.

Saving session states to an external location does not completely guarantee their preservation in case of a server failure. For example, if a server fails while it is modifying the state of a session, some information is lost and subsequent processing using that session can be affected. However, this situation represents a very small period of time when there is a risk of losing session information.

The drawback to saving session states in an external store is that accessing the session state in an external location can use valuable system resources. Session Management can improve system performance by caching the session data at the server level. Multiple consecutive requests that are directed to the same server can find the required state data in the cache, reducing the number of times that the actual session state is accessed in external store and consequently reducing the overhead associated with external location access.

Configuring session management by level

When you configure session management at the Web container level, all applications and the respective Web modules in the Web container normally inherit that configuration, setting up a basic default configuration for the applications and Web modules below it.

However, you can set up different configurations individually for specific applications and Web modules that vary from the Web container default. These different configurations override the default for these applications and Web modules only.

Note: When you overwrite the default session management settings on the application level, all the Web modules below that application inherit this new setting unless they too are set to overwrite these settings.

1. Open the Administrative console.
2. Select the level that this configuration applies to:
 - For the web container level:
 - a. Click **Servers > Application Servers**.

- b. Select a server from the list of application servers.
- c. Under Additional Properties, click **Web Container**.
- For the enterprise application level:
 - a. Click **Applications > Applications**.
 - b. Select an applications from the list of applications.
- For the Web module level:
 - a. Click **Applications > Enterprise Applications**.
 - b. Select an applications from the list of applications.
 - c. Under Related Items, click **Web Modules**.
 - d. Select a Web module from the list of Web modules defined for this application.
- 3. Under **Additional Properties**, click **Session Management**.
- 4. Make whatever changes you need to manage sessions
- 5. If you are working on the Web module or application level and want these settings to override the inherited Session Management settings, under **General Properties**, select **Overwrite**.
- 6. Click **Apply** and **Save**.

Session tracking options

There are several options for session tracking, depending on what sort of tracking method you want to use:

- Session tracking with cookies
- Session tracking with URL rewriting
- Session tracking with Secure Sockets Layer (SSL) information

Session tracking with cookies

Tracking sessions with cookies is the default. No special programming is required to track sessions with cookies.

Session tracking with URL rewriting

An application that uses URL rewriting to track sessions must adhere to certain programming guidelines. The application developer needs to do the following:

- Program servlets to encode URLs
- Supply a servlet or Java Server Pages (JSP) file as an entry point to the application

Using URL rewriting also requires that you enable URL rewriting in the Session Management facility.

Note: In certain cases, clients cannot accept cookies. Therefore, you cannot use cookies as a session tracking mechanism. Applications can use URL rewriting as a substitute.

Program session servlets to encode URLs

Depending on whether the servlet is returning URLs to the browser or redirecting them, include either `encodeURL()` or `encodeRedirectURL()` in the servlet code. Examples demonstrating what to replace in your current servlet code follow.

Rewrite URLs to return to the browser

Suppose you currently have this statement:

```
out.println("<a href=\"/store/catalog\">catalog<a>");
```

Change the servlet to call the `encodeURL` method before sending the URL to the output stream:

```
out.println("<a href=\"");  
out.println(response.encodeURL ("/store/catalog"));  
out.println("\>catalog</a>");
```

Rewrite URLs to redirect

Suppose you currently have the following statement:

```
response.sendRedirect ("http://myhost/store/catalog");
```

Change the servlet to call the `encodeRedirectURL` method before sending the URL to the output stream:

```
response.sendRedirect (response.encodeRedirectURL ("http://myhost/store/catalog"));
```

The `encodeURL()` and `encodeRedirectURL()` methods are part of the `HttpServletResponse` object. These calls check to see if URL rewriting is configured before encoding the URL. If it is not configured, the calls return the original URL.

If both cookies and URL rewriting are enabled and `response.encodeURL()` or `encodeRedirectURL()` is called, the URL is encoded, even if the browser making the HTTP request processed the session cookie.

You can also configure session support to enable protocol switch rewriting. When this option is enabled, the product encodes the URL with the session ID for switching between HTTP and HTTPS protocols.

Supply a servlet or JSP file as an entry point

The entry point to an application (such as the initial screen presented) may not require the use of sessions. However, if the application in general requires session support (meaning some part of it, such as a servlet, requires session support), then after a session is created, all URLs are encoded to perpetuate the session ID for the servlet (or other application component) requiring the session support.

The following example shows how you can embed Java code within a JSP file:

```
<%  
response.encodeURL ("/store/catalog");  
%>
```

Session tracking with SSL information

No special programming is required to track sessions with Secure Sockets Layer (SSL) information.

To use SSL information, turn on **Enable SSL ID tracking** in the Session Management property page. Because the SSL session ID is negotiated between the Web browser and HTTP server, this ID cannot survive an HTTP server failure. However, the failure of an application server does not affect the SSL session ID if an external HTTP Server is present between WebSphere Application Server and the browser.

SSL tracking is supported for the IBM HTTP Server and iPlanet Web servers only. You can control the lifetime of an SSL session ID by configuring options in the Web server. For example, in the IBM HTTP Server, set the configuration variable `SSLV3TIMEOUT` to provide an adequate lifetime for the SSL session ID. An

interval that is too short can cause a premature termination of a session. Also, some Web browsers might have their own timers that affect the lifetime of the SSL session ID. These Web browsers may not leave the SSL session ID active long enough to serve as a useful mechanism for session tracking. Internal Http Server of WebSphere also supports SSL Tracking.

When using the SSL session ID as the session tracking mechanism in a cloned environment, use either cookies or URL rewriting to maintain session affinity. The cookie or rewritten URL contains session affinity information that enables the Web server to properly route a session back to the same server for each request.

Configuring session tracking

To configure session tracking, complete the following:

1. Go to the appropriate level of Session Management.
2. Specify which session tracking mechanism you want to pass the session ID between the browser and the servlet:
 - To track sessions with cookies, click **Enable Cookies**.
To change the cookie settings, click **Modify**.
 - To track sessions with URL rewriting, click **Enable URL Rewriting**.
If you want to enable protocol switch rewriting, click **Enable protocol switch rewriting**.
 - To track sessions with SSL information, click **Enable SSL ID tracking**.
3. Click **Apply**.
4. Click **Save**.
5. Define the session recovery characteristics.

Serializing access to session data

The Servlet API supports concurrent access to a session in a given server instance. WebSphere Application Server provides an option to prevent the concurrent access to a session in a given server instance so that concurrent modification of a session does not occur in a given server instance. This prevention is achieved by synchronizing the requests based on session. When this feature is turned on, a session is obtained for the request before invoking the servlet and requests are synchronized by locking the session for the servlet execution time. Note that synchronization is based on the memory copy of session. So this feature cannot serialize requests across servers based on session when session affinity fails.

Restriction: Use this feature only when concurrent modification of the same session data is possible and is not desirable by the application. This feature has overhead of serializing the requests based on a session.

Do the following to synchronize session access:

1. Select the level of Session Management on which you want to serialize session access.
2. Under Serialize Session access, click **Allow serial access**.
3. In the Maximum wait time box, type the amount of time, in milliseconds, a servlet waits on a session before continuing execution. The default is 120000 milliseconds or two minutes.

4. Select **Allow access on timeout** if you want the servlet to gain access to the session and continue normal execution even if the session is still locked by another servlet. If you do not select this box, the servlet execution will abort when the session request times out.
5. Click **Apply**.
6. Click **Save**.

Session Management settings

Use this page to manage HTTP session support. This support includes specifying a session tracking mechanism, setting maximum in-memory session count, controlling overflow, and configuring session timeout.

To view this administrative console page, click **Servers > Application Servers > *server_name* > Web Container > Session Management**.

Overwrite Session Management (application and Web module level only)

Specifies whether or not these Session Management settings take precedence over those normally inherited from a higher level for the current application or Web module.

By default, Web modules inherit Session Management settings from the application level above it, and applications inherit Session Management settings from the Web container level above it.

Session tracking mechanism

Specifies a mechanism for HTTP session management.

Mechanism	Function	Default
-----------	----------	---------

Enable SSL ID Tracking

Specifies that session tracking uses Secure Sockets Layer (SSL) information as a session ID. Enabling SSL tracking takes precedence over cookie-based session tracking and URL rewriting. 9600 seconds

There are two parameters available if you enable SSL ID tracking: `SSLV3Timeout` and Secure Association Server (SAS). `SSLV3Timeout` specifies the time interval after which SSL sessions are renegotiated. This is a high setting and modification does not provide any significant impact on performance. The SAS parameter establishes an SSL connection only if it goes out of the Java Virtual Machine (JVM) to another JVM. If all the beans are co-located within the same JVM, the SSL used by SAS does not hinder performance.

These are set by editing the `sas.server.properties` and `sas.client.props` files located in the `product_installation_root\properties` directory, where `product_installation_root` is the directory where WebSphere Application Server is installed.

Enable Cookies

Specifies that session tracking uses cookies to carry session IDs. If cookies are enabled, session tracking recognizes session IDs that arrive as cookies and tries to use cookies for sending session IDs. If cookies are not enabled, session tracking uses URL rewriting instead of cookies (if URL rewriting is enabled).

Enabling cookies takes precedence over URL rewriting. Do not disable cookies in the Session Management facility of the application server that is running the administrative application because this action causes the administrative application not to function after a restart of the server. As an alternative, run the administrative application in a separate process from your applications.

Click **Modify** to change these settings.

Enable URL Rewriting

Specifies that the Session Management facility uses rewritten URLs to carry the session IDs. If URL rewriting is enabled, the Session Management facility recognizes session IDs that arrive in the URL if the `encodeURL` method is called in the servlet.

Enable Protocol Switch Rewriting

Specifies that the session ID is added to a URL when the URL requires a switch from HTTP to HTTPS or from HTTPS to HTTP. If rewriting is enabled, the session ID is required to go between HTTP and HTTPS.

Overflow

Specifies that the number of sessions in memory can exceed the value specified by the `Max In Memory Session Count` property. This option is valid only in nondistributed sessions mode.

Maximum in-memory session count

Specifies the maximum number of sessions to maintain in memory. The meaning differs depending on whether you are using in-memory or distributed sessions. For in-memory sessions, this value specifies the number of

sessions in the base session table. Use the Allow Overflow property to specify whether to limit sessions to this number for the entire Session Management facility or to allow additional sessions to be stored in secondary tables. For distributed sessions, this value specifies the size of the memory cache for sessions. When the session cache has reached its maximum size and a new session is requested, the Session Management facility removes the least recently used session from the cache to make room for the new one.

Session timeout

Specifies how long a session can go unused before it is no longer valid. Specify either Set timeout or No timeout. Specify the value in minutes greater than or equal to two.

The value of this setting is used as a default when the session timeout is not specified in a Web module deployment descriptor. Note that to preserve performance, the invalidation timer is not accurate to the second. When the Write Frequency is time based, ensure that this value is least twice as large as the write interval.

Security integration

Specifies that when security integration is enabled, the Session Management facility associates the identity of users with their HTTP sessions

Serialize session access

Specifies that concurrent session access in a given server is not allowed.

Maximum wait time

Specifies the maximum amount of time a servlet request waits on an HTTP session before continuing execution. This parameter is optional and expressed in seconds. The default is 120, or 2 minutes. Under normal conditions, a servlet request waiting for access to an HTTP session gets notified by the request that currently owns the given HTTP session when the request finishes.

Allow access on timeout

Specifies whether the servlet is executed normally or aborted in the event of a timeout. If this box is checked, the servlet executes normally. If this box is not checked, the servlet execution aborts and error logs are generated.

Cookie settings

Use this page to configure cookie settings for session management.

To view this administrative console page, click **Servers > Application Servers > *server_name* > Web Container > Session Management > Enable Cookies.**

Cookie name

Specifies a unique name for the session management cookie. The servlet specification requires the name JSESSIONID. However, for flexibility this value can be configured.

Secure cookies

Specifies that the session cookies include the secure field. Enabling the feature restricts the exchange of cookies to HTTPS sessions only.

Cookie domain

Specifies the domain field of a session tracking cookie. This value controls whether or not a browser sends a cookie to particular servers. For example, if you specify a particular domain, session cookies are sent to hosts in that domain. The default domain is the server.

Cookie path

Specifies that a cookie is sent to the URL designated in the path. Specify any string representing a path on the server. "/" indicates root directory. Specify a value to restrict the paths to which the cookie will be sent. By restricting paths, you prevent the cookie from going to certain URLs on the server. If you specify the root directory, the cookie is sent no matter which path on the given server is accessed.

Cookie maximum age

Specifies the amount of time that the cookie lives on the client browser. Specify that the cookie lives only as long as the current browser session, or to a maximum age. If you choose the maximum age option, specify the age in seconds. This value corresponds to the Time to Live (TTL) value described in the Cookie specification. Default is the current browser session which is equivalent to setting the value to -1.

Distributed sessions

WebSphere Application Server provides the following session mechanisms in a distributed environment:

- **Database Session persistence**, where sessions are stored in the database specified.
- **Memory-to-memory Session replication**, where sessions are stored in one or more specified WebSphere Application Server instances.

When a session contains attributes that implement `HttpSessionActivationListener`, notification occurs anytime the session is activated (that is, session is read to the memory cache) or passivated (that is, session leaves the memory cache). Passivation can occur because of a server shutdown or when the session memory cache is full and an older session is removed from the memory cache to make room for a newer session. It is not guaranteed that a session is passivated in one application server prior to being activated in another.

Session recovery support

For session recovery support, WebSphere Application Server provides distributed session support in the form of database sessions and memory-to-memory replication. Use session recovery support under the following conditions:

- When the user's session data must be maintained across a server restart
- When the user's session data is too valuable to lose through an unexpected server failure

All the attributes set in a session must implement `java.io.Serializable` if the session requires external storage. In general, consider making all objects held by a session serialized, even if immediate plans do not call for session recovery support. If the Web site grows, and session recovery support becomes necessary, the transition occurs transparently to the application if the sessions only hold serialized objects. If not, a switch to session recovery support requires coding changes to make the session contents serialized.

Distributed Environment settings

Use this page to specify a type for saving a session in a distributed environment. To view this administrative console page, click **Servers > Application Servers > server_name > Web Container > Session Management > Distributed Environment Settings**.

Distributed Sessions

Specifies the type of distributed environment to be used for saving sessions.

None	Specifies that the session management facility discards the session data when the server shuts down.
Database	Specifies that the session management facility stores session information in the data source specified by the data source connection settings. Click Database to change these data source settings.
Memory to Memory Replication	Specifies that the session management facility stores the session information in a data source in memory. The session information is copied to other session management facilities for failure recovery. Click Memory to Memory Replication to specify the replicator to use and to change these memory to memory settings. <i>(For WebSphere Application Server Network Deployment only.)</i>

Configuring for database session persistence

To configure the Session Management facility for database session persistence, complete the following:

1. Define a JDBC provider.
2. Create a DB2 table in the z/OS DB2 database that will be used for session persistence.
3. Create a data source pointing to the z/OS DB2 database containing the DB2 table for session persistence, an existing database, using the JDBC provider that you defined. Note the JNDI name of the data source. Under **Data Sources > datasource_name > Custom Properties**, make sure the correct DB2 database is entered for the value of the **databaseName** property. If necessary, contact your DB2 Database Administrator to verify the correct database name.
4. Go to the appropriate level of Session Management.
5. Click **Distributed Environment Settings**
6. Select and click **Database**.
7. Specify the Data Source JNDI name from step 3step 2.
8. Specify the database user ID and password for accessing the database.
9. Retype the password for confirmation.
10. Configure a table space and page sizes for DB2 session databases.
11. Switch to a multirow schema.
12. Click **OK**.
13. If you want to change the tuning parameters, click **Custom Tuning Parameters** and select a setting or customize one.
14. Click **Apply**.

15. Click **Save**.

Switching to a multirow schema

By default, a single session maps to a single row in the database table used to hold sessions. With this setup, there are hard limits to the amount of user-defined, application-specific data that WebSphere Application Server can access.

1. Modify the Session Management facility properties to switch from single to multirow schema.
2. Manually drop and recreate the database table or delete all the rows in the database table that the product uses to maintain HttpSession objects.

To drop the table:

- a. Determine which data source configuration Session Management is using.
- b. In the data source configuration, look up the database name.
- c. Use the database facilities to connect to the database.
- d. Drop the SESSIONS table.

See the *DB2 UDB for OS/390 and z/OS V7 Administration Guide* for a description of how to drop a DB2 database table.

Creating a DB2 table for session persistence describes how to create a new DB2 database table.

Creating a DB2 table for session persistence

If you are using DB2 for session persistence, a DB2 table, in which session data will be collected, must be created and defined to the application server.

To create a DB2 table for collecting session data, do the following:

1. Have your DB2 Administrator create a DB2 database table for storing your session data. (For more information about creating DB2 databases see the *DB2 UDB for OS/390 and z/OS V7 Administration Guide*.)

The table space in which the database table is created must be defined with row level locking (LOCKSIZE ROW). It should also have a page size that is large enough for the objects that will be stored in the table during a session.

Following is an example of a table space definition with row level locking specified and a buffer pool page size of 32K:

```
CREATE DATABASE database_name
  STOGROUP SYSDEFLT
  CCSID EBCDIC;

CREATE TABLESPACE tablespace_name IN database_name
  USING STOGROUP group_name
  PRIQTY 512
  SECQTY 1024
  LOCKSIZE ROW
  BUFFERPOOL BP32K;
```

The Session Manager will use the DB2 table defined within this table space to process the session data. This table must have the following format:

```
CREATE TABLE database_name.table_name (
  ID          VARCHAR(95) NOT NULL ,
  PROPID     VARCHAR(95) NOT NULL ,
  APPNAME    VARCHAR(64) ,
  LISTENERCNT SMALLINT ,
  LASTACCESS DECIMAL(19,0),
```

```

CREATIONTIME    DECIMAL(19,0),
MAXINACTIVETIME INTEGER ,
USERNAME        VARCHAR(256) ,
SMALL           VARCHAR(3122) FOR BIT DATA ,
MEDIUM          VARCHAR(28869) FOR BIT DATA ,
LARGE           BLOB(2097152),
SESSROW         ROWID NOT NULL GENERATED ALWAYS
)
IN database_name.tablespace_name;

```

Note: The length attributes specified for VARCHAR in this example are not necessarily the values your DB2 Administrator should use for the DB2 table he is creating. See the DB2 SQL Reference for the version of DB2 you will be using for guidance in determining appropriate values for these length attributes for your installation.

A unique index must be created on the ID and PROPID columns of this table. The following is an example of the index definition:

```

CREATE UNIQUE INDEX database_name.index_name.
database_name.table_name
(ID ASC,
PROPID ASC,
APPNAME ASC);

```

Note:

- a. At run time, the Session Manager will access the target table using the identity of the J2EE server in which the owning Web application is deployed. Any Web container that is configured to use persistent sessions should be granted both read and update access to the subject database table.
- b. HTTP session processing uses the index defined using the CREATE INDEX statement to avoid database deadlocks. In some situations, such as when a relatively small table size is defined for the database, DB2 may decide not to use this index. When the index isn't used, database deadlocks can occur. If this situation occurs, see the DB2 Administration Guide for the version of DB2 you are using for recommendations on how to calculate the space required for an index, and adjust the size of the tables you are using accordingly.
- c. It may be necessary to tune DB2 in order to make efficient use of the sessions database table and to avoid deadlocks when accessing it. Your DB2 Administrator should refer to the DB2 Administration Guide for specific information about tuning the version of DB2 you are using.

A large object (LOB) table space must be defined and an auxiliary table must be defined within that table space. The following is an example of the LOB table space definition:

```

CREATE LOB TABLESPACE LOB_tablespace_name IN database_name
BUFFERPOOL BP32K
USING STOGROUP group_name
PRIQTY 512
SECQTY 1024
LOCKSIZE LOB;

CREATE AUX TABLE database_name.aux_table_name
IN database_name.LOB_tablespace_name
STORES database_name.table_name
COLUMN LARGE;

```

An index must be created for this auxiliary table. The following is an example of the index definition:

```
CREATE INDEX database_name.aux_index_name ON  
database_name.aux_table_name;
```

2. Have your DB2 Administrator grant the the z/OS userID, under which the server region is running, the appropriate access to this DB2 table. For example, issue the following command to grant z/OS userID CBASRU1, under which the server region is running, access to the table SESSIONS contained in the database SESSDB:

```
GRANT ALL ON SESSDB.SESSIONS TO CBASRU1;
```

3. Use the administrative console to add the name of this DB2 table to the Web container's configuration properties:
 - a. Open the administrative console.
 - b. Click **Servers > Application Servers**.
 - c. Select a server from the list of application servers.
 - d. Under Additional Properties, click **Web Container**.
 - e. Under Additional Properties, click **Custom Properties**.
 - f. Check **SessionTableName** and then click **New**.
 - g. In the **Value** field, enter the name of the DB2 Session Table if you are not using the default value **SESSION**. The name must be in the form *database_name.table_name*. For example, if the database name is SESSDB and the table name is SESSIONS, enter SESSDB.SESSIONS for **Value**.
Optionally, you can update the description of this table in the **Description** field. For example, you might enter "Table name for HTTP session data."
 - h. Click **Apply > Save**.

When the product is restarted, the Session Management facility creates the new SESSIONS table in the specified tablespace.

Database settings

Use this page to specify the settings for database session support.

To view this administrative console page, click **Servers > Application Servers > server_name > Web Container > Session Management > Distributed Environment Settings > Database**.

Datasource JNDI Name

Specifies the datasource description

The JNDI name of the non-XA enabled data source from which Session Management obtains database connections. For example, if the JNDI name of the datasource is "jdbc/sessions", specify "jdbc/sessions." The data source represents a pool of database connections and a configuration for that pool (such as the pool size). The data source must already exist as a configured resource in the environment.

User ID

Specifies the user ID for database access

Password

Specifies the password for database access

Confirm Password

Specifies the password a second time to ensure it recorded correctly.

DB2 Row Size

Specifies the tablespace page size configured for the sessions table, if using a DB2 database. Possible values are 4, 8, 16, and 32 kilobytes (K). The default row size is 4K.

The default row size is 4K. In DB2, it can be updated to a larger value. This can help database performance in some environments. When this value is other than 4, you must specify Table Space Name to use. For 4K pages, the Table Space Name is optional.

Table Space Name

Specifies that tablespace to be used for the sessions table.

This value is required when the DB2 Page Size is other than 4K.

Use Multirow Sessions

Specifies that each instance of application data be placed in a separate row in the database, allowing larger amounts of data to be stored for each session. This action can yield better performance in certain usage scenarios. If using multirow schema is not enabled, instances of application data can be placed in the same row.

Multirow schema considerations

IBM WebSphere Application Server supports the use of a multirow schema option in which each piece of application specific data is stored in a separate row of the database. With this setup, the total amount of data you can place in a session is now bound only by the database capacities. The only practical limit that remains is the size of the session attribute object.

The multirow schema potentially has performance benefits in certain usage scenarios, such as when larger amounts of data are stored in the session but only small amounts are specifically accessed during a given servlet processing of an HTTP request. In such a scenario, avoiding unneeded Java object serialization is beneficial to performance.

Understand that switching between multirow and single row is not a trivial proposition.

In addition to allowing larger session records, using multirow schema can yield performance benefits. However, it requires a little work to switch from single-row to multirow schema, as shown in the instructions below.

Coding considerations and test environment

Consider configuring direct single-row usage to one database and multirow usage to another database while you verify which option suits your application needs. (Do this in code by switching the data source used; then monitor performance.)

Programming issue	Application scenario
Reasons to use single-row	<ul style="list-style-type: none">You can read or write all values with just one record read and write.This takes up less space in a database because you are guaranteed that each session is only one record long.
Reasons not to use single-row	2-megabyte limit of stored data per session.

Programming issue	Application scenario
Reasons to use multirow	<ul style="list-style-type: none"> • The application can store an unlimited amount of data; that is, you are limited only by the size of the database and a 2-megabyte-per-record limit. • The application can read individual fields instead of the whole record. When large amounts of data are stored in the session but only small amounts are specifically accessed during servlet processing of an HTTP request, multirow sessions can improve performance by avoiding unneeded Java object serialization.
Reasons not to use multirow	If data is small in size, you probably do not want the extra overhead of multiple row reads when you can store everything in one row.

In the case of multirow usage, design your application data objects not to have references to each other, to prevent circular references. For example, suppose you are storing two objects A and B in the session using `HttpSession.put(..)` method, and A contains a reference to B. In the multirow case, because objects are stored in different rows of the database, when objects A and B are retrieved later, the object graph between A and B is different than stored. A and B behave as independent objects.

Memory to memory replication

WebSphere Application Server supports session replication to another WebSphere Application Server instance. This support is referred to as *memory to memory session replication*. In this mode, sessions can be replicated to one or more WebSphere Application Server instances to address a single-point of failure (SPOF).

The WebSphere Application Server instance in which the session is currently processed is referred to as the *owner of the session*. In a clustered environment, session affinity in the WebSphere Application Server plug-in routes the requests for a given session to the same server. If the current owner server instance of the session fails, then the WebSphere Application Server plug-in will route the requests to another appropriate server in the cluster. This server either retrieves the session from a server that has the backup copy of the session or it retrieves the session from its own backup copy store, and the session resides there. The server now becomes the owner of the session and affinity is now maintained to this server.

When a session is created or updated in a WebSphere Application Server instance, the session is associated with one of the replicator entries under the replication domain that is configured with the session management facility. This session gets replicated to the WebSphere Application Server instances that are also connected to the same replicator domain and are acting either in "server" or "both" mode (in other words, they are not just replicated "clients").

A WebSphere Application Server instance can be set up to either:

- Only store backup copies of other WebSphere Application Server sessions and not send out copies of any session created in that particular server
- Only broadcast or send out copies of the sessions it owns and not receive back up copies of session from other servers

- Simultaneously broadcast or send out copies of the sessions it owns and act as a backup store for sessions owned by other WebSphere Application Server instances

When configuring the session management facility for memory-to-memory replication, you can choose session management as Server mode, Client Mode, or Both respectively. The default is Both. This storage option is controlled by the mode parameter.

Memory to memory configuration is achieved through the following configurations:

- Peer to peer with a local replicator
- Peer to peer with remote replicators
- Client/server with remote replicators
- Client/server with isolated replicators

In a cluster, by default, sessions are replicated in all the servers in the cluster that are connected to the same replicator domain. This replication can be redundant if there are large number of servers in a cluster. The Session Management facility has an option to partition the servers into groups when storing sessions.

Memory to memory topology: Peer to peer with local replicator

The basic peer to peer (both client and server function, or "both mode") topology is the default configuration. This configuration is based on the peer to peer function with a local replicator (for example, one on the same server) and no partitioning.

In this basic peer to peer topology, each server JVM will:

- Host the web application leveraging HttpSession
- Send out changes to the HttpSessions that it owns
- Receive backup copies of the HttpSessions from all other servers in the cluster
- House a replicator to which the session manager connects

This configuration is the most consolidated topology, where the various system parts are co-located and requires the least amount of server processes. When using this configuration, the most stable implementation is achieved when each of the nodes have equal capabilities (CPU, memory, etc.), and each handles the same amount of work.

This topology is the most redundant since everyone replicates to everyone. This means that on any failure recovery scenario, the server routed to will already have a copy of the session. But as you add servers, more overhead (both CPU and memory) is needed to deal with replication. Scaling will break down the quickest with this configuration because of the machine characteristics, the client hit rate, network speed, and the size on average of the HttpSession. The remaining topologies: client server with isolated replicators, client/server with remote replicators, peer to peer with remote replicators address the issue of scaling.

Configuring for memory to memory replication

To configure the Session Management facility for memory-to-memory replication, complete the following:

1. Create a replication domain and replicator.
2. Go to the appropriate level of Session Management.
3. Click **Distributed Environment Settings**
4. If no replicator has been created for any server, a note appears stating this. In this situation, click **Memory to Memory Replication** to cause the Internal Replication Domain panels to appear. These panels will lead you through the process of creating a replication domain and a replicator. After you have created a replication domain and a replicator, begin again with step 2.
5. If a replicator has been created and associated with this or another server, select **Memory to Memory Replication** and click **Apply**. If a replicator is associated with this server, that replicator is used. If a replicator is not associated with this server, a replicator associated with another server is used.
6. If you want to change replicator for memory-to-memory replication, click **Memory to Memory Replication**, select a replication domain and a replicator from the lists, and specify the Listen to partition IDs. If the replicator is on a different cell, specify an IP address, port details, Listen to partition IDs. If you want to change the runtime mode, select the appropriate setting. When you have finished changing the replicator, click **Apply**
7. Click **OK**.
8. If you want to change the tuning parameters, click **Custom Tuning Parameters** and select a setting or customize one.
9. Click **Apply**.
10. Click **Save**.

Memory-to-memory sessions settings

Use this page to configure memory-to-memory sessions.

To view the Memory-to-memory Sessions page, click **Servers > Application Servers > *server_name* > Web Container > Session Management > Distributed Environment Settings > Memory to Memory Replication**.

Replication

Use one of these options to select the replicator for the session management facility to use for memory-to-memory replication.

Select replicator from the following domain

Specifies a replicator from a replication domain.

Click this option, and select a domain. Then click **Apply** to show all the replicators in that domain. Select the replicator in that domain.

Select replicator from another domain

Specifies a replicator in another domain that is not managed by this cell.

Click this option, and then type the IP address and the port number of the replicator.

Listen to partition groups

Specifies partitions on which the session management facility listens when running in Server mode for memory-to-memory session replication. The number of partitions is defined in the Manage Internal Replication pages. For Server mode and Both mode, the default is all partitions.

Runtime mode

Select the mode in which this server has to run: Both, Client and Server. The mode implies whether data is only sent (client), only received (server), or both. The default is both.

Clustered session support

A clustered environment supports load balancing, where the workload is distributed among the application servers that compose the cluster. In a cluster environment, the same Web application must exist on each of the servers that can access the session. You can accomplish this setup by installing an application onto a cluster definition. Each of the servers in the group can then access the Web application

In a clustered environment, the Session Management facility requires an affinity mechanism so that all requests for a particular session are directed to the same application server instance in the cluster. This requirement conforms to the Servlet 2.3 specification in that multiple requests for a session cannot coexist in multiple application servers. One such solution provided by IBM WebSphere Application Server is *session affinity* in a cluster; this solution is available as part of the WebSphere Application Server plug-ins for Web servers. It also provides for better performance because the sessions are cached in memory. In clustered environments other than WebSphere Application Server clusters, you must use an affinity mechanism (for example, IBM WebSphere Edge Server affinity).

If one of the servers in the cluster fails, it is possible for the request to reroute to another server in the cluster. If distributed sessions support is enabled, the new server can access session data from the database or another WebSphere Application Server instance. You can retrieve the session data only if a new server has access to an external location from which it can retrieve the session.

Tuning session management

IBM WebSphere Application Server session support has features for tuning session performance and operating characteristics, particularly when sessions are configured in a distributed environment. These options support the administrator flexibility in determining the performance and failover characteristics for their environment.

The table summarizes the features, including whether they apply to sessions tracked in memory, in a database, with memory-to-memory replication, or all. Click a feature for details about the feature. Some features are easily manipulated using administrative settings; others require code or database changes.

Feature or option	Goal	Applies to sessions in memory, database, or memory-to-memory
Write frequency	Minimize database write operations.	Database and Memory-to-Memory

Feature or option	Goal	Applies to sessions in memory, database, or memory-to-memory
Session affinity	Access the session in the same application server instance.	All
Multirow schema	Fully utilize database capacities.	Database
Base in-memory session pool size	Fully utilize system capacity without overburdening system.	All
Write contents	Allow flexibility in determining what session data to write	Database and Memory-to-Memory
Scheduled invalidation	Minimize contention between session requests and invalidation of sessions by the Session Management facility. Minimize write operations to database for updates to last access time only.	Database and Memory-to-Memory
Tablespace and row size	Increase efficiency of write operations to database.	Database (DB2 only)

Configuring scheduled invalidation

Instead of relying on the periodic invalidation timer that runs on an interval based on the Session Timeout parameter, you can set specific times for the Session Management facility to scan for invalidated sessions in a distributed environment. When used with distributed sessions, this feature has the following benefits:

- You can schedule the scan for invalidated sessions for times of low application server activity, avoiding contention between invalidation scans of database or another WebSphere Application Server instance and read and write operations to service HTTP session requests.
- Significantly fewer external write operations can occur when running with the End of Service Method write mode because the last access time of the session does not need to be written out on each HTTP request. (Manual Update and Time Based Write options already minimize the writing of the last access time.)

Usage considerations

- With scheduled invalidation configured, HttpSession timeouts are not strictly enforced. Instead, all invalidation processing is handled at the configured invalidation times.
- HttpSessionBindingListener processing is handled at the configured invalidation times unless the HttpSession.invalidate() method is explicitly called.
- The HttpSession.invalidate() method immediately invalidates the session from both the session cache and the external store.

Configuring write contents

In Session Management, you can configure which session data is written to the database or to another WebSphere instance, depending on whether you are using database persistent sessions or memory to memory replication. This flexibility

allows for fewer code changes for the JSP writer when the application will be operating in a clustered environment. The following options are available in Session Management for tuning what is to be written back:

- Write changed (the default) - Write only session data properties that have been updated through `setAttribute()` and `removeAttribute()` method calls.
- Write all - Write all session data properties.

The **Write all** setting might benefit servlet and JSP writers who change Java objects' states that reside as attributes in `HttpSession` and do not call `HttpSession.setAttribute()`.

However, the use of **Write all** could result in more data being written back than is necessary. If this situation applies to you, consider combining the use of **Write all** with **Time-based write** to boost performance overall. As always, be sure to evaluate the advantages and disadvantages for your installation.

With either Write Contents setting, when a session is first created, complete session information is written, including all of the objects bound to the session. When using database session persistence, in subsequent session requests, what is written to the database depends on whether a single-row or multirow schema has been set for the session database, as follows:

Write Contents setting	Behavior with single-row schema	Behavior with multirow schema
Write changed	If any session attribute is updated, all objects bound to the session are written.	Only the session data modified through <code>setAttribute()</code> or <code>removeAttribute()</code> calls is written.
Write all	All bound session attributes are written.	All session attributes that currently reside in the cache are written. If the session has never left the cache, all session attributes are written.

1. Go to the appropriate level of Session Management.
2. Click Distributed Environment Settings
3. Click Custom Tuning Parameters.
4. Select Custom Settings, and click Modify.
5. Select the appropriate write contents setting.

Configuring write frequency

In the Session Management facility, you can configure the frequency for writing session data to the database or to a WebSphere instance, depending on whether you use database distributed sessions or memory-to-memory replication. This flexibility enables you to weigh session performance gains against varying degrees of failover support. The following options are available in the Session Management facility for tuning write frequency:

- `END_OF_SERVICE` - Write session data at the end of the servlet `service()` method call.
- `MANUAL_UPDATE` - Write session data only when the servlet calls the `IBMSession.sync()` method.
- `TIME_BASED_WRITE` (the default) - Write session data at periodic intervals, in seconds (called the *write interval*).

When a session is first created, session information is always written at the end of the `service()` call.

Base in-memory session pool size

The base in-memory session pool size number has different meanings, depending on session support configuration:

- With in-memory sessions, session access is optimized for up to this number of sessions.
- With distributed sessions (meaning, when sessions are stored in a database or in another WebSphere Application Server instance); it also specifies the cache size and the number of last access time updates saved in manual update mode.

For distributed sessions, when the session cache has reached its maximum size and a new session is requested, the Session Management facility removes the least recently used session from the cache to make room for the new one.

General memory requirements for the hardware system, and the usage characteristics of the e-business site, determines the optimum value.

Note that increasing the base in-memory session pool size can necessitate increasing the heap sizes of the Java processes for the corresponding WebSphere Application Servers.

Overflow in nondistributed sessions

By default, the number of sessions maintained in memory is specified by base in-memory session pool size. If you do not wish to place a limit on the number of sessions maintained in memory and allow overflow, set `overflow` to *true*.

Allowing an unlimited amount of sessions can potentially exhaust system memory and even allow for system sabotage. Someone could write a malicious program that continually hits your site and creates sessions, but ignores any cookies or encoded URLs and never utilizes the same session from one HTTP request to the next.

When overflow is disallowed, the Session Management facility still returns a session with the `HttpServletRequest getSession(true)` method when the memory limit is reached, and this is an invalid session that is not saved.

With the WebSphere Application Server extension to `HttpSession`, `com.ibm.websphere.servlet.session.IBMSession`, an `isOverflow()` method returns *true* if the session is such an invalid session. An application can check this status and react accordingly.

Controlling write operations

You can manually control when modified session data is written out to the database or to another WebSphere Application Server instance by using the `sync()` method in the `com.ibm.websphere.servlet.session.IBMSession` interface, which extends the `javax.servlet.http.HttpSession` interface. By calling the `sync()` method from the `service()` method of a servlet, you send any changes in the session to the external location. When `MANUAL_UPDATE` is selected as the write frequency mode, session data changes are written to an external location only if the application calls the `sync()` method. If the `sync()` method is not called, session data changes are lost when a session object leaves the server cache. With

END_OF_SERVICE or TIME_BASE_WRITE is the write frequency mode, the session data changes are written out whenever the sync() method is called. If the sync() method is not called, changes are written out at the end of service method or on a time interval basis based on the write frequency mode selected.

```

IBMSession iSession = (IBMSession) request.getSession();
iSession.setAttribute("name", "Bob");

//force write to external store
iSession.sync( )

```

Tuning parameter settings

Use this page to set tuning parameters for distributed sessions.

To view this administrative console page, click **Servers > Application Servers > *server_name* > Web Container > Session Management > Distributed Environment Settings > Custom Tuning Parameters**.

Tuning Level

Specifies that the session management facility provides certain predefined settings that affect performance.

Select one of these predefined settings or customize a setting. To customize a setting, select one of the predefined settings that comes closest to the setting desired, click **Custom settings**, make your changes, and then click **OK**.

Very high (optimize for performance)

Write frequency	Time based
Write interval	300 seconds
Write contents	Only updated attributes
Schedule sessions cleanup	true
First time of day default	0
Second time of day default	2

High

Write frequency	Time based
Write interval	300 seconds
Write Contents	All session attributes
Schedule sessions cleanup	false

Medium

Write frequency	End of servlet service
Write Contents	Only updated attributes
Schedule sessions cleanup	false

Low (optimize for failover)

Write frequency	End of servlet service
Write Contents	All session attributes
Schedule sessions cleanup	false

Custom settings

Write frequency default	Time based
-------------------------	------------

Write interval default	10 seconds
Write contents default	All session attributes
Schedule sessions cleanup default	false

Tuning parameter custom settings

Use this page to customize tuning parameters for distributed sessions.

To view this administrative console page, click **Servers > Application Servers > *server_name* > Web Container > Session Management > Distributed Environment Settings > Custom Tuning Parameters > Custom settings.**

Write frequency:

Specifies when the session is written to the persistent store.

End of servlet service	A session writes to a database or another WebSphere Application Server instance after the servlet completes execution.
Manual update	A programmatic sync on the IBMSession object is required to write the session data to the database or another WebSphere Application Server instance.
Time based	Session data writes to the database or another WebSphere Application Server instance based on the specified Write Interval value. Default: 10 seconds

Write contents:

Specifies whether updated attributes are only written to the external location or all of the session attributes are written to the external location, regardless of whether or not they changed. The external location can be either a database or another application server instance.

Only updated attributes	Only updated attributes are written to the persistent store.
All session attribute	All attributes are written to the persistent store.

Schedule sessions cleanup:

Specifies when to clean the invalid sessions from a database or another application server instance.

Specify distributed sessions cleanup schedule

Enables the scheduled invalidation process for cleaning up the invalidated HTTP sessions from the external location. Enable this option to reduce the number of updates to a database or another application server instance required to keep the HTTP sessions alive. When this option is not enabled, the invalidator process runs every few minutes to remove invalidated HTTP sessions.

When this option is enabled, specify the two hours of a day for the process to clean up the invalidated sessions in the external location. Specify the times when there is the least activity in the application servers. An external location can be either a database or another application server instance.

First Time of Day (0 - 23)

Indicates the first hour, in Greenwich Mean Time (GMT), during which the invalidated sessions are cleared from the external location. Specify this value as a positive integer between 0 and 23. This value is valid only when schedule invalidation is enabled.

Second Time of Day (0 - 23)

Indicates the second hour, in Greenwich Mean Time (GMT), during which the invalidated sessions are cleared from the external location. Specify this value as a positive integer between 0 and 23. This value is valid only when schedule invalidation is enabled.

Tuning parameter custom settings

Use this page to customize tuning parameters for distributed sessions.

To view this administrative console page, click **Servers > Application Servers > *server_name* > Web Container > Session Management > Distributed Environment Settings > Custom Tuning Parameters > Custom settings**.

Write frequency

Specifies when the session is written to the persistent store.

End of servlet service

A session writes to a database or another WebSphere Application Server instance after the servlet completes execution.

Manual update

A programmatic sync on the `IBMSession` object is required to write the session data to the database or another WebSphere Application Server instance.

Time based

Session data writes to the database or another WebSphere Application Server instance based on the specified Write Interval value. Default: 10 seconds

Write contents

Specifies whether updated attributes are only written to the external location or all of the session attributes are written to the external location, regardless of whether or not they changed. The external location can be either a database or another application server instance.

Only updated attributes

Only updated attributes are written to the persistent store.

All session attribute

All attributes are written to the persistent store.

Schedule sessions cleanup

Specifies when to clean the invalid sessions from a database or another application server instance.

Specify distributed sessions cleanup schedule

Enables the scheduled invalidation process for cleaning up the invalidated HTTP sessions from the external location. Enable this option to reduce the number of updates to a database or another application server instance required to keep the HTTP sessions alive. When this option is not enabled, the invalidator process runs every few minutes to remove invalidated HTTP sessions.

When this option is enabled, specify the two hours of a day for the process to clean up the invalidated sessions in the external location. Specify the times when there is the least activity in the application servers. An external location can be either a database or another application server instance.

First Time of Day (0 - 23)

Indicates the first hour, in Greenwich Mean Time (GMT), during which the invalidated sessions are cleared from the external location. Specify this value as a positive integer between 0 and 23. This value is valid only when schedule invalidation is enabled.

Second Time of Day (0 - 23)

Indicates the second hour, in Greenwich Mean Time (GMT), during which the invalidated sessions are cleared from the external location. Specify this value as a positive integer between 0 and 23. This value is valid only when schedule invalidation is enabled.

Best practices for using HTTP Sessions

- **Enable Security integration for securing HTTP sessions**

HTTP sessions are identified by session IDs. A session ID is a pseudo-random number generated at the runtime. Session hijacking is a known attack HTTP sessions and can be prevented if all the requests going over the network are enforced to be over a secure connection (meaning, HTTPS). But not every configuration in a customer environment enforces this constraint because of the performance impact of SSL connections. Due to this relaxed mode, HTTP session is vulnerable to hijacking and because of this vulnerability, WebSphere Application Server has the option to tightly integrate HTTP sessions and WebSphere Application Server security. Enable security in WebSphere Application Server so that, the sessions are protected in a manner that only users who created the sessions are allowed to access them.

- **Release HttpSession objects using `javax.servlet.http.HttpSession.invalidate()` when finished.**

HttpSession objects live inside the Web container until:

- The application explicitly and programmatically releases it using the `javax.servlet.http.HttpSession.invalidate()` method; quite often, programmatic invalidation is part of an application logout function.
- WebSphere Application Server destroys the allocated `HttpSession` when it expires (default = 1800 seconds or 30 minutes). The WebSphere Application Server can only maintain a certain number of HTTP sessions in memory based on Session Management settings. In case of distributed sessions, when maximum cache limit is reached in memory, the Session Management facility removes the least recently used (LRU) one from cache to make room for a session.

- **Avoid trying to save and reuse the `HttpSession` object outside of each servlet or JSP file.**

The `HttpSession` object is a function of the `HttpServletRequest` (you can get it only through the `req.getSession()` method), and a copy of it is valid only for the life of the `service()` method of the servlet or JSP file. You *cannot* cache the `HttpSession` object and refer to it outside the scope of a servlet or JSP file.

- **Implement the `Serializable` class when developing new objects to be stored in the HTTP session.**

This action allows the object to properly serialize when using distributed sessions. Without this extension, the object cannot serialize correctly and throws an error. An example of this follows:

```
public class MyObject implements java.io.Serializable {...}
```

Make sure all instance variable objects that are not marked `transient` are serializable.

- **The `HttpSession` API does not dictate transactional behavior for sessions.**

Distributed `HttpSession` support does not guarantee transactional integrity of an attribute in a failover scenario or when session affinity is broken. Use transactionally aware resources like enterprise Java beans to guarantee the transaction integrity required by your application.

- **Ensure the Java objects you add to a session are in the correct class path.**

If you add Java objects to a session, place the class files for those objects in the correct classpath (the Application Classpath if utilizing sharing across Web modules in an enterprise application, or the WebModule Classpath if using the Servlet 2.2-complaint session sharing) or in the directory containing other servlets used in WebSphere Application Server. In the case of session clustering, this action applies to every node in the cluster.

Because the `HttpSession` object is shared among servlets that the user might access, consider adopting a site-wide naming convention to avoid conflicts.

- **Avoid storing large object graphs in the `HttpSession` object.**

In most applications each servlet only requires a fraction of the total session data. However, by storing the data in the `HttpSession` object as one large object, an application forces WebSphere Application Server to process all of it each time.

- **Utilize Session Affinity to help achieve higher cache hits in the WebSphere Application Server.**

WebSphere Application Server has functionality in the HTTP Server plug-in to help with session affinity. The plug-in will read the cookie data (or encoded URL) from the browser and helps direct the request to the appropriate application or clone based on the assigned session key. This functionality increases use of the in-memory cache and reduces hits to the database or another WebSphere Application Server instance

- **Maximize use of session affinity and avoid breaking affinity.**

Using session affinity properly can enhance the performance of the WebSphere Application Server. Session affinity in the WebSphere Application Server environment is a way to maximize the in-memory cache of session objects and reduce the amount of reads to the database or another WebSphere Application Server instance. Session affinity works by caching the session objects in the server instance of the application with which a user is interacting. If the application is deployed in multiple servers of a server group, the application can direct the user to any one of the servers. If the users starts on server1 and then comes in on server2 a little later, the server must write all of the session information to the external location so that the server instance in which server2 is running can read the database. You can avoid this database read using session affinity. With session affinity, the user starts on server1 for the first request; then for every successive request, the user is directed back to server1. Server1 has to look only at the cache to get the session information; server1 never has to make a call to the session database to get the information.

You can improve performance by not breaking session affinity. Some suggestions to help avoid breaking session affinity are:

- Combine all Web applications into a single application server instance, if possible, and use modeling or cloning to provide failover support.
- Create the session for the frame page, but do not create sessions for the pages within the frame when using multiframe JSP files. (See discussion later in this topic.)
- **When using multi-framed pages, follow these guidelines:**
 - Create a session in only one frame or before accessing any frame sets. For example, assuming there is no session already associated with the browser and a user accesses a multi-framed JSP file, the browser issues concurrent requests for the JSP files. Because the requests are not part of any session, the JSP files end up creating multiple sessions and all of the cookies are sent back to the browser. The browser honors only the last cookie that arrives. Therefore, only the client can retrieve the session associated with the last cookie. Creating a session before accessing multi-framed pages that utilize JSP files is recommended.
 - By default, JSPs get a HttpSession using `request.getSession(true)` method. So by default JSPs create a new session if none exists for the client. Each JSP page in the browser is requesting a new session, but only one session is used per browser instance. A developer can use `<% @ page session="false" %>` to turn off the automatic session creation from the JSP files that will not access the session. Then if the page needs access to the session information, the developer can use `<%HttpSession session = javax.servlet.http.HttpServletRequest.getSession(false); %>` to get the already existing session that was created by the original session creating JSP file. This action helps prevent breaking session affinity on the initial loading of the frame pages.
 - Update session data using only one frame. When using framesets, requests come into the HTTP server concurrently. Modifying session data within only one frame so that session changes are not overwritten by session changes in concurrent frameset is recommended.
 - Avoid using multi-framed JSP files where the frames point to different Web applications. This action results in losing the session created by another Web application because the JSESSIONID cookie from the first Web application gets overwritten by the JSESSIONID created by the second Web application.
- **Secure all of the pages (not just some) when applying security to servlets or JSP files that use sessions with security integration enabled, .**

When it comes to security and sessions, it is all or nothing. It does not make sense to protect access to session state only part of the time. When security

integration is enabled in the Session Management facility, all resources from which a session is created or accessed must be either secured or unsecured. You cannot mix secured and unsecured resources.

The problem with securing only a couple of pages is that sessions created in secured pages are created under the identity of the authenticated user. Only the same user can access sessions in other secured pages. To protect these sessions from use by unauthorized users, you cannot access these sessions from an unsecured page. When a request from an unsecured page occurs, access is denied and an `UnauthorizedSessionRequestException` error is thrown. (`UnauthorizedSessionRequestException` is a runtime exception; it is logged for you.)

- **Use manual update and either the `sync()` method or time-based write in applications that read session data, and update infrequently.**

With `END_OF_SERVICE` as write frequency, when an application uses sessions and anytime data is read from or written to that session, the `LastAccess` time field updates. If database sessions are used, a new write to the database is produced. This activity is a performance hit that you can avoid using the Manual Update option and having the record written back to the database only when data values update, not on every read or write of the record.

To use manual update, turn it on in the Session Management Service. (See the tables above for location information.) Additionally, the application code must use the `com.ibm.websphere.servlet.session.IBMSession` class instead of the generic `HttpSession`. Within the `IBMSession` object there is a method called `sync()`. This method tells the WebSphere Application Server to write the data in the session object to the database. This activity helps the developer to improve overall performance by having the session information persist only when necessary.

Note: An alternative to using the manual updates is to utilize the timed updates to persist data at different time intervals. This action provides similar results as the manual update scheme.

- Implement the following suggestions to achieve high performance:
 - If your applications do not change the session data frequently, use Manual Update and the `sync()` function (or timed interval update) to efficiently persist session information.
 - Keep the amount of data stored in the session as small as possible. With the ease of using sessions to hold data, sometimes too much data is stored in the session objects. Determine a proper balance of data storage and performance to effectively use sessions.
 - If using database sessions, use a dedicated database for the session database. Avoid using the application database. This helps to avoid contention for JDBC connections and allows for better database performance.
 - If using memory to memory sessions, define replicators only on the servers and have the client attach to server replicator.
 - If using memory to memory sessions, employ partitioning (either group or single replica) as your clusters grow in size and scaling decreases.
 - Verify that you have the latest e-fixes for the WebSphere Application Server.
- Utilize the following tools to help monitor session performance.
 - Run the `com.ibm.servlet.personalization.sessiontracking.IBMTrackerDebug` servlet. - To run this servlet, you must have the servlet invoker running in the Web application you want to run this from. Or, you can explicitly configure this servlet in the application you want to run.
 - Use the WebSphere Application Server Resource Analyzer which comes with WebSphere Application Server to monitor active sessions and statistics for the WebSphere Application Server environment.

- Use database tracking tools such as "Monitoring" in DB2. (See the respective documentation for the database system used.)

Managing HTTP sessions: Resources for learning:

Use the following links to find relevant supplemental information about HTTP sessions. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

View links to additional information about:

- "Managing HTTP sessions: Resources for learning:"
- "Managing HTTP sessions: Resources for learning:"
- "Managing HTTP sessions: Resources for learning:"

Programming model and decisions

- Best practices
- HTTP Session Persistence Best Practices
- Improving session persistence performance with DB2
- Persistent client state HTTP cookies specification

Programming instructions and examples

- Java Servlet documentation, tutorials, and examples site

Programming specifications

- Java Servlet 2.3 API specification download site
- J2EE 1.3 specification download site

Chapter 4. Using enterprise beans in applications

1. Design a J2EE application and the enterprise beans that it needs. See "Resources for learning" for links to design information that is specific to enterprise beans.
2. Develop any enterprise beans that your application will use.
3. Prepare for assembly. For your EJB 2.x-compliant entity beans, decide on an appropriate access intent policy.
4. Assemble the beans into one or more EJB modules. This includes setting security.
5. Assemble the modules into a J2EE application.
6. Deploy the application in an application server.
7. Test the modules.
 - As needed, debug problems with the container.
 - Debug access and deployment problems.
8. Assemble the production application.
9. Deploy the application to a production environment.
10. Manage the application:
 - a. Manage installed EJB modules. After an application has been installed, you can manage its EJB modules individually through administrative console settings.
 - b. Manage other aspects of the J2EE application.
11. Update the module and redeploy it.
12. Tune the performance of the application. See Best practices for developing enterprise beans.

Enterprise beans

An enterprise bean is a Java component that can be combined with other resources to create J2EE applications. There are three types of enterprise beans, *entity* beans, *session* beans, and *message-driven* beans.

All beans reside in EJB containers, which provide an interface between the beans and the application server on which they reside.

Entity beans store permanent data. Entity beans with container-managed persistence (CMP) require connections to a form of persistent storage. This storage might be a database, an existing legacy application, a file, or other types of persistent storage. Entity beans with bean-managed persistence manage permanent data in whichever manner is defined in the bean code. This can include writing to databases or XML files, for example.

Session beans do not require database access, although they can obtain it indirectly as needed through entity beans. Session beans can also obtain direct access to databases (and other resources) through the use of resource references. Session beans can be either *stateful* or *stateless*.

New in the Enterprise JavaBeans (EJB) specification, version 2.0, message-driven beans enable asynchronous message servicing. The EJB container and a Java Message Service (JMS) provider work together to process messages. When a message arrives from another application component through JMS, the EJB

container forwards it through an `onMessage()` call to a message-driven bean instance, which then processes the message. In other respects, message-driven beans are similar to stateless session beans.

Beans that require data access use *data sources*, which are administrative resources that define pools of connections to persistent storage mechanisms.

For more information about enterprise beans, see "Resources for learning."

Developing enterprise beans

Design a J2EE application and the enterprise beans that it needs.

- For general design information, see "Resources for learning."
- Before developing entity beans with container-managed persistence (CMP), read "Concurrency control."

There are two basic approaches to selecting tools for developing enterprise beans:

- You can use one of the available integrated development environments (IDEs). IDE tools automatically generate significant parts of the enterprise bean code and contain integrated tools for packaging and testing enterprise beans. The IBM WebSphere Application Developer product is the recommended IDE. For more information, see the documentation for that product.
- If you have decided to develop enterprise beans without an IDE, you need at least an ASCII text editor. You can also use a Java development tool that does not support enterprise bean development. You can then use tools available in the Java Software Development Kit (SDK) and in this product to assemble, test, and deploy the beans.

The following steps primarily support the second approach, development without an IDE.

1. If necessary, migrate any pre-existing code to the required version of the Enterprise JavaBeans (EJB) specification.
2. Write and compile the components of the enterprise bean.
 - At a minimum, an EJB 1.1 session bean requires a bean class, a home interface, and a remote interface. An EJB 1.1 entity bean requires a bean class, a primary-key class, a home interface, and a remote interface.
 - At a minimum, an EJB 2.0 session bean requires a bean class, a home or local home interface, and a remote or local interface. An EJB 2.0 entity bean requires a bean class, a primary-key class, a remote home or local home interface, and a remote or local interface. The types of interfaces go together: If you implement a local interface, you must define a local home interface as well.
 - Available only through EJB 2.0, a message-driven bean requires only a bean class.
3. For each entity bean, complete work to handle persistence operations.
 - Create a database schema for the entity bean's persistent data.
 - For entity beans with container-managed persistence(CMP), you must store the bean's persistent data in one of the supported databases. The Application Assembly Tool automatically generates SQL code for creating database tables for CMP entity beans. If your CMP beans require complex database mappings, it is recommended that you use the IBM WebSphere Studio Application Developer product to generate code for the database tables.

- For entity beans with bean-managed persistence (BMP), you can create the database and database table by using the database tools or use an existing database and database table.

For more information on creating databases and database tables, consult your database documentation.

- **(CMP entity beans for EJB 2.0 only)** Define finder queries with EJB Query Language (EJB QL).

With EJB QL, you define finders in terms of CMP fields and container-managed relationships, as follows:

- *Public* finders are visible in the bean's home interface. Implemented in the bean class, they return only remote interfaces and collection types.
- *Private* finders, expressed as SELECT statements, are used only within the bean class. They can return both local and remote interfaces, dependent values, other CMP field types, and collection types.
- **(CMP entity beans for EJB 1.1 only: an IBM extension)** Create a finder helper interface for each CMP entity bean that contains specialized finder methods (other than the `findByPrimaryKey` method).

The following logic is required for each finder method (other than the `findByPrimaryKey` method) contained in the home interface of an entity bean with CMP:

- The logic must be defined in a public interface named *NameBeanFinderHelper*, where *Name* is the name of the enterprise bean (for example, `AccountBeanFinderHelper`).
- The logic must be contained in a String constant named *findMethodName* where *Clause*, where *findMethodName* is the name of the finder method. The String constant can contain zero or more question marks (?) that are replaced from left to right with the value of the finder method's arguments when that method is called.

Assemble the beans in one or more EJB modules.

Migrating enterprise bean code to the supported specification

Support for Version 2.0 of the Enterprise JavaBeans (EJB) specification is new for Version 5 of this product. Migration of enterprise beans deployed in Version 4.0.x of this product is not generally necessary; Version 1.1 of the EJB specification is still supported. Follow these steps as appropriate for your application deployment.

1. Modify enterprise bean code for changes in the specification.
 - For Version 1.0 beans, migrate at least to Version 1.1.
 - As stated previously, migration from Version 1.1 to Version 2.0 of the EJB specification is not required for redeployment on this version of the product. However, if your application requires the capabilities of Version 2.0, migrate your Version 1.1-compliant code.

Note: The EJB Version 2.0 specification mandates that prior to the EJB container's executing a `findByMethod` query, the state of all enterprise beans enlisted in the current transaction be synchronized with the persistent store. (This is so the query is performed against current data.) If Version 1.1 beans are reassembled into an EJB 2.0-compliant module, the EJB container synchronizes the state of Version 1.1 beans as well as that of Version 2.0 beans. As a result, you might notice some change in application behavior even though the application code for the Version 1.1 beans has not been changed.

2. Modify enterprise bean code for changes in deployment requirements. If the enterprise beans were previously deployed in Version 3.0.x of this product, modify import statements to match standard package names. In Version 3.0.2.x, the following standard packages were present under nonstandard names:

```
javax.sql.*  
javax.transaction.*
```

Any code using WebSphere data sources, including BMP entity beans and session beans that access databases, must be modified.

3. You might have to modify code for some EJB 1.1-compliant modules that were not migrated to Version 2.0. Use the following information to help you decide.
 - Some stub classes for deployed enterprise beans have changed in the Java 2 Software Development Kit, Version 1.4.1.
 - The task of generating deployment code for enterprise beans changed significantly for EJB 1.1-compliant modules relative to EJB 1.0-compliant modules.
 - If the CMP beans write to databases with mixed-case table or column names and you used IBM VisualAge for Java, Version 3.5.x, to generate the original deployment code, you cannot simply reassemble the beans in this product. You must export the original EJB project from the VisualAge for Java product as an EJB 1.1 JAR. This preserves the metadata needed to generate the correct deployment code for mixed-case database tables and columns. For more information, see the documentation for the Deployment Tool for Enterprise JavaBeans.

For detailed information about source and binary compatibility between deployed versions, see "Resources for learning."

4. Reassemble and redeploy all modules to incorporate migrated code.

Migrating enterprise bean code from Version 1.0 to Version 1.1

The following information generally applies to any enterprise bean that currently complies with Version 1.0 of the Enterprise JavaBeans (EJB) specification. For more information about migrating code for beans produced with the IBM WebSphere Studio Application Developer tool, see the documentation for that product. For more information about migrating code in general, see "Resources for learning."

1. In session beans, replace all uses of `javax.jts.UserTransaction` with `javax.transaction.UserTransaction`. Entity beans may no longer use the `UserTransaction` interface at all.
2. In finder methods for entity beans, include `FinderException` in the throws clause.
3. Remove throws of `java.rmi.RemoteException`; throw `javax.ejb.EJBException` instead. However, continue to include `RemoteException` in the throws clause of home and remote interfaces as required by the use of Remote Method Invocation (RMI).
4. Remove uses of the `finalize()` method.
5. Replace calls to `getCallerIdentity()` with calls to `getCallerPrincipal()`. The use of `getCallerIdentity()` is deprecated.
6. Replace calls to `isCallerInRole(Identity)` with calls to `isCallerinRole (String)`. The use of `isCallerInRole(Identity)` and `java.security.Identity` is deprecated.
7. Replace calls to `getEnvironment()` in favor of JNDI lookup. As an example, change the following code:

```
String homeName =
    aLink.getEntityContext().getEnvironment().getProperty("TARGET_HOME_NAME");
if (homeName == null) homeName = "TARGET_HOME_NAME";
```

The updated code would look something like the following:

```
Context env = (Context)(new InitialContext()).lookup("java:comp/env");
String homeName = (String)env.lookup("ejb10-properties/TARGET_HOME_NAME");
```

8. In `ejbCreate` methods for an entity bean with container-managed persistence (CMP), return the bean's primary key class instead of `void`.
9. Add the `getHomeHandle()` method to home interfaces.

```
public javax.ejb.HomeHandle getHomeHandle() {return null;}
```

Consider enhancements to match the following changes in the specification:

- Primary keys for entity beans can be of type `java.lang.String`.
- Finder methods for entity beans return `java.util.Collection`.
- Check the format of any JNDI names being used. Local name spaces are also supported.
- Security is defined by role, and isolation levels are defined at the method level rather than at the bean level.

Migrating enterprise bean code from Version 1.1 to Version 2.0

Enterprise JavaBeans (EJB) Version 2.0-compliant beans may be assembled only in an EJB 2.0-compliant module, although an EJB 2.0-compliant module can contain a mixture of Version 1.x and Version 2.0 beans.

The EJB Version 2.0 specification mandates that prior to the EJB container's executing a `findByMethod` query, the state of all enterprise beans enlisted in the current transaction be synchronized with the persistent store. (This is so the query is performed against current data.) If Version 1.1 beans are reassembled into an EJB 2.0-compliant module, the EJB container synchronizes the state of Version 1.1 beans as well as that of Version 2.0 beans. As a result, you might notice some change in application behavior even though the application code for the Version 1.1 beans has not been changed.

The following information generally applies to any enterprise bean that currently complies with Version 1.1 of the EJB specification. For more information about migrating code for beans produced with the IBM WebSphere Studio Application Developer tool, see the documentation for that product. For more information about migrating code in general, see "Resources for learning."

1. In beans with container-managed persistence (CMP) version 1.x, replace each CMP field with abstract get and set methods. In doing so, you must make each bean class abstract.
2. In beans with CMP version 1.x, change all occurrences of `this.field = value` to `setField(value)`.
3. In each CMP bean, create abstract get and set methods for the primary key.
4. In beans with CMP version 1.x, create an EJB Query Language statement for each finder method.
5. In finder methods for beans with CMP version 1.x, return `java.util.Collection` instead of `java.util.Enumeration`.
6. Update handling of non-application exceptions.
 - To report non-application exceptions, throw `javax.ejb.EJBException` instead of `java.rmi.RemoteException`.

- Modify rollback behavior as needed: In EJB versions 1.1 and 2.0, all non-application exceptions thrown by the bean instance result in the rollback of the transaction in which the instance is running; the instance is discarded. In EJB 1.0, the container does not roll back the transaction or discard the instance if it throws `java.rmi.RemoteException`.
7. Update rollback behavior as the result of application exceptions.
- In EJB versions 1.1 and 2.0, an application exception does not cause the EJB container to automatically roll back a transaction.
 - In EJB Version 1.1, the container performs the rollback only if the instance has called `setRollbackOnly()` on its `EJBContext` object.
 - In EJB Version 1.0, the container is required to roll back a transaction when an application exception is passed through a transaction boundary started by the container.

WebSphere extensions to the Enterprise JavaBeans specification

This article outlines extensions to the Enterprise JavaBeans (EJB) specification that IBM provides with this product:

Inheritance in enterprise beans

In the Java language, *inheritance* is the creation of a new class from an existing class or a new interface from an existing interface. This product supports two forms of inheritance: standard class inheritance and EJB inheritance.

In standard class inheritance, the home interface, remote interface, or enterprise bean class inherits properties and methods from base classes that are not themselves enterprise bean classes or interfaces.

By contrast in enterprise bean inheritance, an enterprise bean inherits properties (such as container-managed persistence (CMP) fields and container-managed relationship (CMR) fields), methods, and method-level control descriptor attributes from another enterprise bean.

For more information, see the documentation for the IBM WebSphere Studio Application Developer product.

Optimistic concurrency control for container-managed persistence

This product supports optimistic concurrency control of data access.

Access intents for EJB persistence

Performance enhancements

Through the lifetime-in-cache settings, this product provides a way for you to improve performance for beans that are only occasionally updated. For more information, see "Entity bean assembly settings."

Some enterprise beans created with the IBM WebSphere Studio Application Developer product can utilize *read-ahead* for loading a bean and its related beans in a single database operation. An entire object graph or any part of the graph can be preloaded by configuring a finder method to use read-ahead.

Assembly and deployment extensions

This product supports IBM extensions of assembly and deployment options. IBM extensions are clearly marked in reference topics for assembly settings.

Best practices for developing enterprise beans

Use the following guidelines when designing and developing enterprise beans:

- Use a stateless session bean to act as the entry point for business logic. For more information about using session facades, see "Resources for learning."
- Entity beans should use container-managed persistence.
- In an Enterprise JavaBeans (EJB) Version 2.0 environment, use local interfaces to improve communication between enterprise beans in the same Java virtual machine.

Local calls avoid the overhead of RMI/IIOP and use pass-by-reference semantics instead of pass-by-value. For each call, the caller and callee beans share the state of arguments. EJB 2.0 beans can have both a local and remote interface but more typically have one or the other.

- For communicating with remote clients, provide remote and remote home interfaces. For communicating with local clients like servlets, entity beans, and message-driven beans, provide local and local home interfaces.

Using access intent policies

Apply access intent policies to methods of CMP entity beans.

Access intent policies

An access intent policy is a named set of properties (access intents) that governs data access for Enterprise JavaBeans (EJB) persistence. You can assign policies to an entity bean and to individual methods on an entity bean's home, remote, or local interfaces during assembly. Access intents are settable only within EJB Version 2.x-compliant modules for entity beans with CMP Version 2.x.

This product supplies a number of access intent policies that specify permutations of read intent and concurrency control; the pessimistic/update policy can be qualified further. The selected policy determines the appropriate isolation level and locking strategy used by the run-time environment.

Access intent policies are specifically designed to supplant the use of isolation level and access intent method-level modifiers found in the extended deployment descriptor for EJB version 1.1 enterprise beans. You cannot specify isolation level and read-only modifiers for EJB version 2.0 enterprise beans.

Access intent policies configured on an entity basis define the default access intent for that entity. The default access intent is used to control that entity in the absence of a more specific configuration based upon either method-level policy configuration or application profiling.

For entity beans that are backed by tables with nullable columns, use an optimistic policy with caution. Nullable columns are automatically excluded from overqualified updates at deployment time; concurrent changes to a nullable field might result in lost updates. When used with the IBM WebSphere Studio Application Developer product, this product provides support for selecting a subset of the nonnullable columns that are to be reflected in the overqualified update statement that is generated in the deployment code to support optimistic policies.

Concurrency control

Concurrency control is the management of contention for data resources. A concurrency control scheme is considered *pessimistic* when it locks a given resource early in the data-access transaction and does not release it until the transaction is closed. A concurrency control scheme is considered *optimistic* when locks are acquired and released over a very short period of time at the end of a transaction. The objective of optimistic concurrency is to minimize the time over which a given resource would be unavailable for use by other transactions. This is especially important with long-running transactions, which under a pessimistic scheme would lock up a resource for unacceptably long periods of time.

Under an optimistic scheme, locks are obtained immediately before a read operation and released immediately afterwards. Update locks are obtained immediately before an update operation and held until the end of the transaction.

To enable optimistic concurrency, this product uses an *overqualified update scheme* to test whether the underlying data source has been updated by another transaction since the beginning of the current transaction. With this scheme, the columns marked for update and their original values are added explicitly through a WHERE clause in the UPDATE statement so that the statement fails if the underlying column values have been changed. As a result, this scheme can provide column-level concurrency control; pessimistic schemes can control concurrency at the row level only.

Optimistic schemes typically perform this type of test only at the end of a transaction. If the underlying columns have not been updated since the beginning of the transaction, pending updates to container-managed persistence fields are committed and the locks are released. If locks cannot be acquired or if some other transaction has updated the columns since the beginning of the current transaction, the transaction is rolled back: All work performed within the transaction is lost.

Pessimistic and optimistic concurrency schemes require different transaction isolation levels. Enterprise beans that participate in the same transaction and require different concurrency control schemes cannot operate on the same underlying data connection.

Whether or not to use optimistic concurrency depends on the type of transaction. Transactions with a high penalty for failure might be better managed with a pessimistic scheme. (A high-penalty transaction is one for which recovery would be risky or resource-intensive.) For low-penalty transactions, it is often worth the risk of failure to gain efficiency through the use of an optimistic scheme. In general, optimistic concurrency is more efficient when update collisions are expected to be infrequent; pessimistic concurrency is more efficient when update collisions are expected to occur often.

Read-ahead hints

Read-ahead schemes enable applications to minimize the number of database roundtrips by retrieving a working set of container-managed persistence (CMP) beans for the transaction within one query. Read-ahead involves activating the requested CMP beans and caching the data for their related beans, which ensures that data is present for the beans that are most likely to be needed next by an application. A *read-ahead hint* is a canonical representation of the related beans that are to be read. It is associated with the *findByPrimaryKey* method for the requested bean type, which must be an EJB 2.x-compliant CMP entity bean.

A read-ahead hint takes the form of a character string. You do not have to provide the string; the wizard generates it for you based on CMRs defined for the bean. The following example is provided as supplemental information only.

Suppose a CMP bean type A has a finder method that returns instances of bean A. A read-ahead hint for this method is specified using the following notation:
RelB.RelC; RelD

Interpret the preceding notation as follows:

- Bean type A has a CMR with bean types B and D.
- Bean type B has a CMR with bean type C.

For each bean of type A that is retrieved from the database, its directly-related B and D beans and its indirectly-related C beans are also retrieved. The order of the retrieved bean data columns in each row of the result set is the same as their order in the read-ahead hint: an A bean, a B bean (or null), a C bean (or null), a D bean (or null). For hints in which the same relationship is mentioned more than once (for example, *RelB.RelC;RelB.RelE*), a bean's data columns appear only once, at the position it first appears in the hint.

The tokens shown in the notation (*RelB* and so on) must be CMR field names for the relationships as defined in the deployment descriptor for the bean. In indirect relationships such as *RelB.RelC*, *RelC* is a CMR field name defined in the deployment descriptor for bean type B.

A single read-ahead hint cannot refer to the same bean type in more than one relationship. For example, if a Department bean has a relationship *employees* with the Employee bean and also has a relationship *manager* with the Employee bean, the read-ahead hint cannot specify both *employees* and *manager*.

For more information about how to set read-ahead hints, see the documentation for the Websphere Studio Application Developer product.

Applying access intent policies to methods

You apply an access intent policy to a method, or set of methods, in an application's entity beans through the Application Assembly Tool (AAT).

1. Start the AAT.
2. Create or edit the application EAR file. For example, to change attributes of an existing application, select **File > Open**, then select the EAR file.
3. Select **EJB Modules > moduleName > Access Intent**.
4. To configure a new access intent policy, right-click and select **New**.
5. On the **New Access Intent** panel, specify a name and a description. These attributes are provided as a convenience to the developer and are not used at run time.
6. To select the methods to which the access intent policy should apply, click **Add** beside the Methods table.
7. From the **Applied access intent** list, select an access intent policy.
8. To override an attribute defined in the applied policy, click **Add** beside the Access intent attribute overrides table.
9. Click **OK** to exit the New Access Intent panel.
10. Save your configuration by selecting **File > Save**.

Access intent exceptions

The following exceptions are thrown in response to the application of access intent policies:

com.ibm.ws.ejbpersistence.utilpm.PersistenceManagerException

If the method that drives the `ejbLoad()` method is configured to be read-only but updates are then made within the transaction that loaded the bean's state, an exception is thrown during invocation of the `ejbStore()` method, and the transaction is rolled back. Likewise, the `ejbRemove()` method cannot succeed in a transaction that is set as read-only. If an update hint is applied to methods of entity beans with bean-managed persistence, the same behavior and exception results. The forwarded exception object contains the message string `PMGR1103E: update instance level read only bean beanName`

This exception is also thrown if the applied access intent policy cannot be honored because a finder, `ejbSelect`, or container-managed relationship (CMR) accessor method returns an inherently read-only result. The forwarded exception object contains the message string `PMGR1001: No such DataAccessSpec - methodName`

The most common occurrence of this error is when a custom finder that contains a read-only EJB Query Language (EJB QL) statement is called with an applied access intent of `wsPessimisticUpdate` or `wsPessimisticUpdate-Exclusive`. These policies require the use of a FOR UPDATE clause on the SQL SELECT statement to be executed, but a read-only query cannot support FOR UPDATE. Other examples of read-only queries include joins; the use of ORDER BY, GROUP BY, and DISTINCT keywords.

To eliminate the exception, edit the EJB query so that it does not return an inherently read-only result or change the access intent policy being applied.

- If an update access is required, change the applied access intent setting to `wsPessimisticUpdate-WeakestLockAtLoad` or `wsOptimisticUpdate`.
- If update access is not truly required, use `wsPessimisticRead` or `wsOptimisticRead`.
- If connection sharing between entity beans is required, use `wsPessimisticUpdate-WeakestLockAtLoad` or `wsPessimisticRead`.

com.ibm.websphere.ejb.container.CollectionCannotBeFurtherAccessed

If a lazy collection is driven after it is no longer in scope, and beyond what has already been locally buffered, a `CollectionCannotBeFurtherAccessed` exception is thrown.

com.ibm.ws.exception.RuntimeWarning

If an application is configured incorrectly, a run-time warning exception is thrown as the application starts; startup is ended. You can validate an application's configuration by choosing the verify function in the WebSphere Application Assembly Tool. Some examples of misconfiguration include:

- A method configured with two different access intent policies
- A method configured with an undefined access intent policy

Access intent assembly settings

Access intent policies contain data-access settings for use by the persistence manager. Default access intent policies are configured on the entity bean. Optionally, you can associate access intent policies with one or more methods.

These settings are applicable only for EJB 2.x-compliant entity beans that are packaged in EJB 2.x-compliant modules. Connection sharing between beans with

bean-managed persistence and those with container-managed persistence is possible if they all use the same access intent policy.

Name

Specifies a name for a mapping between an access intent policy and one or more methods.

Description

Contains text that describes the mapping.

Methods - Name

Specifies the name of an enterprise bean method, or the asterisk character (*). The asterisk is used to denote all of the methods of an enterprise bean's remote and home interfaces.

Methods - Enterprise bean

Specifies which enterprise bean contains the methods indicated in the Name setting.

Methods - Type

Used to distinguish between a method with the same signature that is defined in both the home and remote interface. Use Unspecified if an access intent policy applies to all methods of the bean.

Data type

String

Range

Valid values are Home, Remote, Local, LocalHome or Unspecified

Methods - Parameters

Contains a list of fully qualified Java type names of the method parameters. This setting is used to identify a single method among multiple methods with an overloaded method name.

Applied access intent

Specifies how the container must manage data access for persistence. Configurable both as a default access intent for an entity and as part of a method-level access intent policy.

Data type

String

Range

Valid settings are wsPessimisticUpdate, wsPessimisticUpdate-NoCollision, wsPessimisticUpdate-Exclusive, wsPessimisticUpdate-WeakestLockAtLoad, wsPessimisticRead, wsOptimisticUpdate, or wsOptimisticRead. Only wsPessimisticRead and wsOptimisticRead are valid when class-level caching is enabled in the EJB container.

This product supports lazy collections. For each segment of a collection, iterating through the collection (*next()*) does not trigger a remote method call to retrieve the next remote reference. Two policies (*wsPessimisticUpdate* and *wsPessimisticUpdate-Exclusive*) are extremely lazy; the collection increment size is set to 1 to avoid overlocking the application. The other policies have a collection increment size of 25.

Additional information about valid settings follows:

Profile name	Concurrency control	Access type	Transaction isolation
wsPessimisticRead (Note 1)	pessimistic	read	For Oracle, read committed. Otherwise, repeatable read
wsPessimisticUpdate (Note 2)	pessimistic	update	For Oracle, read committed. Otherwise, repeatable read
wsPessimisticUpdate-Exclusive (Note 3)	pessimistic	update	serializable
wsPessimisticUpdate-NoCollision (Note 4)	pessimistic	update	read committed
wsPessimisticUpdate-WeakestLockAtLoad (Note 5)	pessimistic	update	Repeatable read
wsOptimisticRead	optimistic	read	read committed
wsOptimisticUpdate (Note 6)	optimistic	update	read committed
Notes: <ol style="list-style-type: none"> 1. Read locks are held for the duration of the transaction. 2. The generated SELECT FOR UPDATE query grabs locks at the beginning of the transaction. 3. SELECT FOR UPDATE is generated; locks are held for the duration of the transaction. 4. Generated overqualified-update query forces failure if CMP column values have changed since the beginning of the transaction. 			

Access intent best practices

This topic outlines issues to consider when applying access intent policies to Enterprise JavaBeans (EJB) methods.

- **Take care when applying wsPessimisticUpdate-NoCollision.** This policy does not ensure data integrity. No database locks are held, so concurrent transactions can overwrite each other's updates. Use this policy only if you can be sure that only one transaction will attempt to update persistent store at any given time.

Frequently asked questions: Access intent

I have not applied any access intent policies at all. My application runs just fine with a DB2 database, but it fails with an Oracle database with the following message: com.ibm.ws.ejbpersistence.utilpm.PersistenceManagerException: PMGR1001E: No such DataAccessSpec :FindAllCustomers. The backend datastore does not support the SQLStatement needed by this AccessIntent: (pessimistic update-weakestLockAtLoad)(collections: transaction/25) (resource manager prefetch: 0) (AccessIntentImpl@d23690a). Why?

If you have not configured access intent, all of your data is accessed under the default access intent policy (wsPessimisticUpdate-WeakestLockAtLoad). On DB2 databases, the weakest lock is a shared one, and the query runs without a FOR UPDATE clause. On Oracle databases, however, the weakest lock is an update lock; this means that the SQL query must contain a FOR UPDATE clause. However, not every SQL statement necessarily supports FOR UPDATE; for example, if the query is being run against multiple tables in a join, FOR UPDATE is not supported.

To avoid this problem, try either of the following:

- Modify your SQL query or reconfigure your application so that an update lock is supported
- Apply an access intent policy that supports optimistic concurrency

I have two beans in a container-managed relationship. I call `findByPrimaryKey()` on the first bean and then call `getBean2()`, a CMR accessor method, on the returned instance. At that point, I get an `InconsistentAccessIntentException`.

Why? You are probably using read-ahead. When you loaded the first bean, you caused the second bean to be loaded under the access intent policy applied to the finder method for the first bean. However, you have configured your CMR accessor method from the first bean to the second with a different access intent policy. CMR accessor methods are really finder methods in disguise; the run-time environment behaves as if you were trying to change the access intent for an instance you have already read from persistent store.

To avoid this problem, beans configured in a read-ahead hint are all driven to load with the same access intent policy as the bean to which the read-ahead hint is applied.

I have a bean with a one-to-many relationship to a second bean. The first bean has a pessimistic-update intent policy applied. When I try to add an instance of the second bean to the first bean's collection, I get an

`UpdateCannotProceedWithIntegrityException`. Why?

The second bean probably has a read intent policy applied. When you add the second bean to the first bean's collection, you are not updating the first bean's state, you are implicitly modifying the second bean's state. (The second bean contains a foreign key to the first bean, which is modified.)

To avoid this problem, ensure that both ends of the relationship have an update intent policy applied if you expect to change the relationship at run time.

EJB modules

An EJB module is used to assemble one or more enterprise beans into a single deployable unit. An EJB module is stored in a standard Java archive (JAR) file.

An EJB module contains the following:

- One or more deployable enterprise beans.
- A deployment descriptor, stored in an Extensible Markup Language (XML) file. This file declares the contents of the module, defines the structure and external dependencies of the beans in the module, and describes how the beans are to be used at run time.

An EJB module can be used as a stand-alone application, or it can be combined with other EJB modules, or with Web modules, to create a J2EE application. An EJB module is installed and run in an enterprise bean container.

For more information about EJB modules, see "Resources for learning."

Assembling EJB modules

If you want to use existing Java 2 Platform, Enterprise Edition (J2EE) Version 1.2 modules in your J2EE Version 1.3 application, migrate them to the Version 1.3 specification first.

Assemble an Enterprise JavaBeans (EJB) module to contain enterprise beans and related code artifacts. Group Web components, client code, and resource adapter code in separate modules.

An EJB module is installed as a stand-alone application or is combined with other modules into an enterprise application.

To increase performance, break CMP enterprise beans into several enterprise bean modules during assembly. The load time for hundreds of beans is improved by distributing the beans across several JAR files and packaging them to an EAR file. Load time is faster when the administrative server attempts to start the beans, for example, 8-10 minutes versus more than one hour when one JAR file is used.

The Application Assembly Tool (AAT) provides flexibility in assembling EJB modules. Options described below include:

- Importing an existing EJB module (EJB JAR file)
 - Creating a new EJB module
 - Copying code artifacts (such as entity beans) from one EJB module into a new EJB module
1. Start the AAT.
 2. From the New tab, select **EJB Module**. Click **OK**. The navigation tree displays various sets of properties for configuring the new EJB module.
 3. Use the property dialog shown in the AAT workspace to change the default file name and location.
 - a. It is recommended that you change the display name so that it differs from the file name.
 - b. If you like, change the temporary location of the EJB module from the default location, *install_root/bin*.
 4. Add at least one EJB component to the module.
 - Add at least one enterprise bean to the EJB component.
 - Import an existing JAR or EAR file containing EJB components.
 - a. In the Navigation pane, right-click the **EJB Components** icon.
 - b. Select **Import** from the pop-up menu.
 - c. Click **Browse** to locate the archive file to import.
 - d. Click **Open** to display the contents of the archive file. The applications in the selected archive file display.
 - e. Select an EJB application from the archive file.
 - f. Select the servlets or JSP files to be added, and click **Add** to display the components in the Selected Components window.
 - g. Click **OK** to add the selected components.
 - Copy and paste values from an existing module.
 - Create a new EJB component.
 - a. In the Navigation pane, right-click the **EJB Components** icon.
 - b. Select **New** from the pop-up menu.
 - c. Enter the component name and archive type.
 - d. Select the class files.
 - e. Click **OK** in the **New EJB Component** property dialog.
 - f. Enter properties for the EJB component as needed.
 5. Enter assembly properties for each bean.
 - a. Click the plus sign (+) next to the component instance to show property groups.
 - b. Right-click the icon for a property group.
 - c. Select **New** from the pop-up menu to add new values, or edit existing values in the property pane.

If you change the session type (stateful or stateless) of a session bean, you must click **Apply** before making any other changes to the same bean. Otherwise, certain input fields on the GUI become inactive. (You will know they are inactive because they are grayed out on the GUI.) After making all of your changes, click **Apply** again to commit them.

6. Add any other files needed by the application.
 - a. Right-click the **Files** icon.
 - b. Select **Add Files** from the pop-up menu.
 - c. Select **Browse** to navigate the directory structure.
 - d. Click **Select** to open an archive.
 - e. Select the files to add and click **Add**.
 - f. In the Selected Files window, click **OK** to add the files.

Assemble any other new modules of your choice:

- EJB modules
- Application client modules
- Resource adapter modules

You can also migrate existing modules.

Another option is to proceed directly to assembling a new application module. While assembling an application module, you can create any new modules that you need.

CMP field assembly settings

In Enterprise JavaBeans (EJB) Version 1.1-compliant beans, container-managed persistence (CMP) fields define the variables in the bean class for which the container must handle persistence management. In EJB Version 2.0-compliant beans, these are replaced by abstract get and set methods; generated code provides the implementation of these abstract methods.

Name

Specifies a subset of public variables in the enterprise bean's implementation class.

Container transactions

Container transaction properties specify how an EJB container is to manage transaction scopes for the enterprise bean's method invocations. A transaction attribute is mapped to one or more methods.

Container transaction assembly settings

Container transaction settings specify how an EJB container is to manage transaction scopes for the enterprise bean's method invocations. Specify one or more methods and associate a transaction attribute with each method.

Name

Specifies a name for the mapping between a transaction attribute and one or more methods.

Description

Contains text that describes the mapping.

Transaction attribute

Specifies how the container must manage the transaction boundaries when delegating a method invocation to an enterprise bean's business method.

Data type	String
Default	Required
Range	For all but message-driven beans, valid values are Mandatory, Never, Not Supported, Required, Requires New, Supports. For session beans, Bean Managed is also valid. For message-driven beans, only Bean Managed, Not Supported, and Required are valid.

Additional information about valid values follows:

Bean Managed

Notifies the container that the bean class directly handles transaction demarcation. This setting can be specified for session beans and (in EJB 2.0 implementations only) for message-driven beans, and it cannot be specified for individual bean methods.

Mandatory

Directs the container to always call the bean method within the transaction context associated with the client. If the client attempts to invoke the bean method without a transaction context, the container throws the `javax.jts.TransactionRequiredException` exception to the client. The transaction context is passed to any EJB object or resource accessed by an enterprise bean method.

EJB clients that access these entity beans must do so within an existing transaction. For other enterprise beans, the enterprise bean or bean method must implement the Bean Managed value or use the Required or Requires New value. For non-enterprise bean EJB clients, the client must access a transaction by using the `javax.transaction.UserTransaction` interface.

Never Directs the container to invoke bean methods without a transaction context.

- If the client calls a bean method from within a transaction context, the container throws the `java.rmi.RemoteException` exception.
- If the client calls a bean method from outside a transaction context, the container behaves in the same way as if the Not Supported transaction attribute was set. The client must call the method without a transaction context.

Not Supported

Directs the container to call the bean method without a transaction context. If a client calls a bean method from within a transaction context, the container suspends the association between the transaction and the current thread before invoking the method on the enterprise bean instance. The container then resumes the suspended association when the method invocation returns. The suspended transaction context is not passed to any enterprise bean objects or resources that are used by this bean method.

Required

Directs the container to call the bean method within a transaction context. If a client calls a bean method from within a transaction context, the container calls the bean method within the client transaction context. If a client calls a bean method outside a transaction context, the container creates a new transaction context and calls the bean method from within that context. The transaction context is passed to any enterprise bean objects or resources that are used by this bean method.

Requires New

Directs the container to always call the bean method within a new transaction context, regardless of whether the client calls the method

within or outside a transaction context. The transaction context is passed to any enterprise bean objects or resources that are used by this bean method.

Supports

Directs the container to call the bean method within a transaction context if the client calls the bean method within a transaction. If the client calls the bean method without a transaction context, the container calls the bean method without a transaction context. The transaction context is passed to any enterprise bean objects or resources that are used by this bean method.

Methods - Name

Specifies the name of an enterprise bean method, or the asterisk character (*). The asterisk is used to denote all methods of an enterprise bean's remote and home interfaces.

Methods - Enterprise bean

Specifies which enterprise bean contains the methods indicated in the **Name** setting.

Methods - Type

Used to distinguish between a method with the same signature that is defined in both the home and remote interface. Use `Unspecified` if a transaction attribute applies to all methods of the bean.

Data type

String

Range

Valid values for EJB 1.1 implementations are `Home`, `Remote`, or `Unspecified`. For EJB 2.0 implementations, `Local` and `LocalHome` are also valid.

Methods - Parameters

Contains a list of fully qualified Java type names of the method parameters. This setting is used to identify a single method among multiple methods with an overloaded method name.

EJB module assembly settings

An EJB module is used to assemble enterprise beans into a single deployable unit. An EJB module contains one or more enterprise beans and a deployment descriptor.

File name

Specifies the file name of the EJB module, relative to the top level of the application package.

Alternate DD

Specifies a deployment descriptor to be used at run time instead of the one installed in the module.

Classpath

The path that contains additional classes required by the application that are not contained in the module's archive file. The class loader uses this path. Specify the values relative to the root of the EAR file and separate the values with spaces. Absolute values that refer to files or directories on the hard drive are ignored. To specify classes that are not in JAR files but are in the root of the EAR file, use a period and forward slash (`./`). Consider the following example directory structure in which the file `myapp.ear` contains an EJB module named `myejb.jar`. Additional classes reside in `class1.jar` and `class2.zip`. A class named `xyz.class` is not packaged in a JAR file but is in the root of the EAR file.

myapp.ear/myejb.jar
myapp.ear/class1.jar
myapp.ear/class2.zip
myapp.ear/xyz.class

Specify `class1.jar class2.zip ./` as the value of the Classpath setting. (Name only the directory for .class files.)

Display name

Specifies a short name that is intended to be displayed by GUIs.

Description

Contains text that describes the module.

EJB client JAR

Specifies the location of a JAR file that contains a subset of deployed classes needed by the client.

Small icon

Specifies the name of a JPEG or GIF file that contains a small image (16x16 pixels). The image is used as an icon to represent the module in a GUI.

Large icon

Specifies the name of a JPEG or GIF file that contains a large image (32x32 pixels). The image is used as an icon to represent the module in a GUI.

Generalizations - Subtype

Information about this property is not available.

This property is an IBM extension to the standard J2EE deployment descriptor.

Generalizations - Supertype

Information about this property is not available.

This property is an IBM extension to the standard J2EE deployment descriptor.

EJB relationships - Name

The logical name for a container-managed relationship between EJB 2.0-compliant entity beans.

Default data source - JNDI name

Specifies the default JNDI name for the data source. This default is used if binding information is not specified in the deployment descriptor for an individual enterprise bean.

Default CMP connection factory

Specifies the JNDI name for a CMP connection factory. This setting is applicable only for EJB 2.x-compliant CMP beans.

Default authorization - User ID

Specifies the default user ID for connecting to an enterprise bean's data store.

Default authorization - Password

Specifies the default password for connecting to an enterprise bean's data store.

Entity bean assembly settings

An entity bean encapsulates persistent data, which is stored in a data source, and associated methods to manipulate that data.

EJB name

Specifies a logical name for the enterprise bean. This name must be unique within the EJB module. There is no relationship between this name and the JNDI name.

Display name

Specifies a short name that is intended to be displayed by GUIs.

Description

Contains text that describes the entity bean.

EJB class

Specifies the full name of the enterprise bean class (for example, `com.ibm.ejs.doc.account.AccountBean`).

Remote - Home

(Required for EJB 1.x) Specifies the full name of the enterprise bean's home interface class (for example, `com.ibm.ejs.doc.account.AccountHome`).

Remote - Interface

(Required for EJB 1.x) Specifies the full name of the enterprise bean's remote interface class (for example, `com.ibm.ejs.doc.account.Account`).

Local interface - Home

(Required for EJB 1.x) Specifies the full name of the enterprise bean's local home interface class (for example, `com.ibm.ejs.doc.account.AccountLocalHome`).

Local interface - Interface

(Required for EJB 1.x) Specifies the full name of the enterprise bean's local interface class (for example, `com.ibm.ejs.doc.account.AccountLocal`).

Persistence type

Specifies whether an entity bean manages its own persistent storage or whether storage is managed by the container.

Data type

String

Range

Valid values are Bean managed and Container managed.

Reentrant

Specifies whether the entity bean is reentrant. If an enterprise bean is reentrant, it can call methods on itself or call another bean that calls a method on the calling bean. Only entity beans can be reentrant.

If an entity bean is not reentrant and a bean instance is executing a client request in a transaction context and another client using the same transaction context makes a request on the same bean instance, the EJB container throws the `java.rmi.RemoteException` exception to the second client. If a bean is reentrant, the container cannot distinguish this type of illegal loopback call from a legal concurrent call, so the bean must be coded to detect illegal loopback calls.

Primary key class

Specifies the full name of the bean's primary key class (for example, `com.ibm.ejs.doc.account.AccountKey`). Composite primary keys map to multiple fields in the entity bean class (or to data structures built from the primitive Java data types) and must be encapsulated in a primary key class.

More complicated enterprise beans are likely to have composite primary keys, with multiple instance variables representing the primary key. A subset of the container-managed fields is used to define the primary key class associated with each instance of an enterprise bean.

Primary key field

Specifies the name of a simple primary key. Simple primary keys map to a single field in the entity bean class and are made up of primitive Java data types (such as integer or long). If exactly one CMP field is the primary key, it can be specified here.

Data type	String
Range	Valid values are the name of any one CMP field or Compound key, which appears when the primary key class is set

Version

Specifies the version of EJB specification with which a container-managed persistence (CMP) entity bean complies.

Data type	String
Range	Valid values are 1.x or 2.x

Abstract schema name

Specifies the name of the abstract schema type of an EJB Version 2.x CMP entity bean. It is used in EJB Query Language (QL) queries. For example, the abstract schema name might be Order for an entity bean whose local interface is com.acme.commerce.Order.

Small icon

Specifies the name of a JPEG or GIF file that contains a small image (16x16 pixels). The image is used as an icon to represent the entity bean in a GUI.

Large icon

Specifies the name of a JPEG or GIF file that contains a large image (32x32 pixels). The image is used as an icon to represent the entity bean in a GUI.

Security identity

Specifies that a principal's credential properties are to be handled as indicated in the Run-As mode property. If this setting is enabled, the Run-As mode property can be edited.

Run-As mode

Specifies the credential information to be used by the security service to determine the permissions that a principal has on various resources.

At appropriate points, the security service determines whether the principal is authorized to use a particular resource based on the principal's permissions. If the method call is authorized, the security service acts on the principal's credential properties according to the Run-As mode setting of the enterprise bean.

Data type	Enumerated integer
Range	Valid values are Use identity of caller and Use identity assigned to specified role

Additional information about valid settings follows:

Use identity of caller

The security service makes no changes to the principal's credential properties.

Use identity assigned to specified role

A principal that has been assigned to the specified security role is used for the execution of the bean's methods. This association is part of the

application binding in which the role is associated with a user ID and password of a user who is granted that role.

Role name

Specifies the name of a security role. If **Run-As mode** is set to Use identity assigned to specified role, a principal that has been granted this role is used.

Description

Contains further information about the security role.

Concurrency control

Specifies how the bean is to handle concurrent access to its data. This setting is applicable only for EJB 1.x-compliant entity beans.

This property is an IBM extension to the standard J2EE deployment descriptor.

Data type	String
Range	Valid values are Optimistic or Pessimistic

Inheritance root

Specifies whether the enterprise bean is at the root of an inheritance hierarchy.

This property is an IBM extension to the standard J2EE deployment descriptor.

Bean Cache - Activate at

Specifies the point at which an enterprise bean is activated and placed in the cache. Removal from the cache and passivation is also governed by this setting.

This property is an IBM extension to the standard J2EE deployment descriptor.

Data type	String
Default	Transaction
Range	Valid values are Once, Transaction, and Activity session

More information about valid values follows:

Once Indicates that the bean activates when it is first accessed in the server process, and passivates (and is removed from the cache) at the discretion of the container, for example, when the cache becomes full.

Transaction

Indicates that the bean activates at the start of a transaction and passivates (and is removed from the cache) at the end of the transaction.

Activity session

Indicates that the bean activates and passivates as follows:

- On an ActivitySession boundary, if an ActivitySession context is present on activation
- On a transaction boundary, if a transaction context (but no ActivitySession context) is present on activation
- Otherwise, on an invocation boundary

The values of the **Activate at** and **Load at** settings govern which commit options are used, as follows:

- For commit option A (implies exclusive database access), use **Activate at** = Once and **Load at** = Activation.

This option reduces database I/O (avoids calls to the `ejbLoad` function) but serializes all transactions accessing the bean instance. Option A can increase memory usage by maintaining more objects in the cache, but can provide better

response time if bean instances are not generally accessed concurrently by multiple transactions. To use Option A successfully, you must also set Concurrency control to Pessimistic.

Note for Network Deployment users

Note: When workload management is enabled, you cannot use Option A. You must use settings that result in the use of options B or C.

- For commit option B (implies shared database access), use `Activate at = Once` and `Load at = Transaction`.

Option B can increase memory usage by maintaining more objects in the cache. However, because each transaction creates its own copy of an object, there can be multiple copies of an instance in memory at any given time (one per transaction), requiring database access at each transaction. If an enterprise bean contains a significant number of calls to the `ejbActivate` function, using Option B is beneficial because the required object is already in the cache. Otherwise, this option does not provide significant benefits over Option A.

- For commit option C (implies shared database access), use `Activate at = Transaction` and `Load at = Transaction`.

This option reduces memory usage by maintaining fewer objects in the cache; however, there can be multiple copies of an instance in memory at any given time (one per transaction). This option can reduce transaction contention for enterprise bean instances that are accessed concurrently but not updated.

Bean Cache - Load at

Specifies when the bean loads its state from the database. The value of this setting implies whether the container has exclusive or shared access to the database. This property is an IBM extension to the standard J2EE deployment descriptor.

Data type	String
Default	Transaction
Range	Valid values are Activation and Transaction

Additional information about valid values follows:

Activation

Indicates that the bean loads when it is activated (regardless of `Activate at` setting) and implies that the container has exclusive access to the database.

Transaction

Indicates that the bean loads at the start of a transaction and implies that the container has shared access to the database.

The `Activate at` and `Load at` settings govern which commit options are used. The commit options are described in the Enterprise JavaBeans specification. For more information about this setting and achieving a given commit behavior, see `Bean Cache - Activate at`.

Commit option

Specifies which commit option is used as a result of bean cache settings. The commit options are described in the Enterprise JavaBeans specification.

Data type	String
Range	Valid values are A, B, and C

Local Transactions - Unresolved action

Specifies the action that the EJB container must take if resources are uncommitted by an application in a local transaction.

This property is an IBM extension to the standard J2EE deployment descriptor. This setting is applicable only when **Resolution control** is set to `Application`. A local transaction context is created when a method runs in what the EJB specification refers to as an unspecified transaction context.

Data type	String
Default	Rollback
Range	Valid values are <code>Commit</code> and <code>Rollback</code>

Additional information about these settings follows:

Commit

At end of the local transaction context, the container instructs all unresolved local transactions to commit.

Rollback

(Default) At end of the local transaction context, the container instructs all unresolved local transactions to roll back.

Local Transactions - Resolution control

Specifies how the local transaction is to be resolved before the local transaction context ends: by the application through user code or by the EJB container. This property is an IBM extension to the standard J2EE deployment descriptor.

Data type	String
Range	Valid values are <code>Application</code> and <code>ContainerAtBoundary</code>

Additional information about these settings follows:

Application

When this setting is used, your code must either commit or roll back the local transaction. If this does not occur, the runtime environment logs a warning and automatically commits or rolls back the connection as specified by the **Unresolved action** setting.

ContainerAtBoundary

When this setting is used, the container takes responsibility for resolving each local transaction. This provides you with a programming model similar to global transactions in which your code simply gets a connection and performs work within it. User code does not have to handle local transactions.

- If the **Boundary** attribute is set to `ActivitySession`, then the local transactions are enlisted as `ActivitySession` resources and directed to complete by the `ActivitySession`.
- If the the **Boundary** attribute is set to `BeanMethod`, then the local transactions are committed at method end by the container.

Connections are never committed automatically by the resource adapter when this value is configured for the bean **Unresolved action** is not used. An application cannot call `Connection.LocalTransaction.begin()` when using this policy and receives an exception from the resource adapter if it does so.

When using a **Resolution control** of `ContainerAtBoundary`, applications must get connection handles *after* the local transaction context boundary has been started by the container. The application should close the connection before the end of the boundary, although any work performed on the connection is not committed or rolled back until the local

transaction context ends. This model of connection usage is sometimes referred to as the “get-use-close” model.

This value is supported only for EJB components that use container-managed transactions. It is not supported for web components or for enterprise beans that use bean-managed transactions.

Local Transactions - Boundary

Specifies the duration of a local transaction context.

This property is an IBM extension to the standard J2EE deployment descriptor.

Data type	String
Default	BeanMethod
Range	Valid values are BeanMethod and ActivitySession

Additional information about valid settings follows:

BeanMethod

When this setting is used, the local transaction begins when the method begins and ends when the method ends.

ActivitySession

When this setting is used, the local transaction must be resolved within the scope of any ActivitySession in which it was started or, if no ActivitySession context is present, within the same bean method in which it was started.

This property can be changed on WebSphere Application Server Enterprise only.

Local Relationship Roles - Name

Within a local relationship between EJB 1.x-compliant entity beans, the logical name for the view an entity bean presents to other beans in the relationship. For example, in a relationship between Account and Customer beans, the role of the Account instance relative to the Customer instance might be *savingsAccount*.

This property is an IBM extension to the standard J2EE deployment descriptor. This is separate from the container-managed relationships defined in the Enterprise JavaBeans specification, Version 2.0.

Local Relationship Roles - Source EJB Name

The name of the entity bean for which the role is defined.

This property is an IBM extension to the standard J2EE deployment descriptor.

Local Relationship Roles - is Forward

Specifies how deployment code for navigating the relationship is generated. This setting is applicable only for navigable relationships.

If **isForward** is enabled (set to true), deployment code is generated in the source bean. That is, navigation of the relationship proceeds *forward* from the source to the target.

Otherwise, deployment code is generated in the target bean. That is, navigation of the relationship proceeds from the target to the source.

This property is an IBM extension to the standard J2EE deployment descriptor. For more information, see the documentation for the Deployment Tool for Enterprise JavaBeans.

Local Relationship Roles - is Navigable

Specifies whether data in related beans may be retrieved through queries to the source bean.

This property is an IBM extension to the standard J2EE deployment descriptor.

Lifetime in cache

The lifetime, in seconds, of cached data for an instance of this bean type.

This value indicates how long the cached data is to exist beyond the end of the transaction in which the data was retrieved. This might avoid another retrieval from persistent storage if the same bean instance were to be used in later transactions. How this value is interpreted depends on the value of **Lifetime in cache usage**.

This property is an IBM extension to the standard J2EE deployment descriptor.

Data type	Long
Units	Seconds
Default	0
Range	0 to 2 ⁶¹ - 1

Lifetime in cache usage

Indicates how the lifetime-in-cache setting is to be used by the caching mechanism. This property is an IBM extension to the standard J2EE deployment descriptor.

If your application uses CMP beans in which the underlying data changes infrequently, you might gain significantly better performance by using this setting with **Lifetime in cache**. Typically, data read from persistent storage is held temporarily in an internal cache until the state of the instance is restored. Cached data normally does not persist beyond state restoration or the end of the transaction in which the finder method was called. By setting **Lifetime in cache usage** to a value other than Off, you indicate that the cached data is to be held for a longer time, potentially hours or days, before invalidating the version of the data in the cache and fetching a new version. Avoiding a trip to persistent storage greatly speeds up access to such beans by applications.

In addition, the use of a value other than Off requires that finders on the bean have an access type of Optimistic Read (if you are only reading instances of the bean) or Optimistic Update (if you plan to occasionally update instances of the bean).

- For EJB 1.x-compliant beans, see Access intent - access type.
- For EJB 2.x-compliant beans, see Applied access intent.

Setting Load at to activation and Activate at to Once also minimizes retrievals from persistent storage. However, this settings combination might not be supported by certain CMP beans because it results in the `ejbLoad()` method being called once instead of at the beginning of each transaction in which they are used. The lifetime-in-cache settings combination is independent of CMP bean implementation, though it does incur the modest overhead of calling `ejbLoad()` on each use.

Data type	Enumerated int
Units	Not applicable
Default	0 (Off)
Range	Valid values are Clock Time, Elapsed Time, Week Time, or Off

Additional information about valid values follows:

Off When this value is used, the value of **Lifetime in cache** is ignored. Beans of this type are cached only in a transaction-scoped cache. The cached data for this instance expires after the transaction in which it was retrieved is completed.

Elapsed Time

When this value is used, the value of **Lifetime in cache** is added to the time at which the transaction in which the bean instance was retrieved is completed. The resulting value becomes the time at which the cached data expires. The value of **Lifetime in cache** can add up to minutes, hours, days, and so on.

Clock Time

When this value is used, the value of **Lifetime in cache** represents a particular time of day. The value is added to the immediately preceding or following midnight to calculate a future time value, which is then treated as for Elapsed Time. Using Clock Time enables you to specify that all instances of this bean type are to have their cached data invalidated at, for example, 3 AM, no matter when they were retrieved. This is important if, for example, the data underlying this bean type is batch-updated at 3 AM every day.

The selection of midnight (preceding or following) depends on the value of **Lifetime in cache**. If **Lifetime in cache** plus the value that represents the preceding midnight is earlier than the current time, the following midnight is used.

When you use Clock Time, the value of **Lifetime in cache** is not supposed to represent more than 24 hours. If it does, the cache manager subtracts 24-hour increments from it until a value less than or equal to 24 hours is achieved. To invalidate data at midnight, set **Lifetime in cache** to 0.

Week Time

Usage of this value is the same as for Clock Time, except that the value of **Lifetime in cache** is added to the preceding or following Sunday midnight (11:59 PM Saturday plus 1 minute). When Week Time is used, the value of **Lifetime in cache** can represent more than 24 hours but not more than 7 days.

Default Access Intent

Specifies the default access intent under which the entity should load.

JNDI name

Specifies the JNDI name of the bean's home interface. This is the name under which the enterprise bean's home interface is registered and therefore, is the name that must be specified when an EJB client does a lookup of the home interface.

Data source - JNDI name

Specifies the JNDI name for the bean's data source.

Default Authorization - User ID

Specifies the default user ID for connecting to a data source.

Default Authorization - Password

Specifies the default password for connecting to a data source.

CMP Resource - JNDI name

Specifies the JNDI name for the resource by which CMP data is stored.

CMP Resource - Resource authentication

Specifies the scope at which resources are to be authenticated: by the container or by the resource.

Message-driven bean assembly settings

Use this page to configure the assembly properties of message-driven beans

For more information about the effect of JMS properties, such as message selectors and message acknowledgement, see the WebSphere MQ *Using Java* book, SC34-5456 or Sun's Java Message Service (JMS) specification documentation

The following notebook pages are available:

Page tab

Description

General properties

Specify general assembly properties for the message bean.

- EJB name
- Display name
- Description
- EJB class
- Transaction type

Advanced properties

Specify advanced assembly properties for the message bean.

- Message selector
- Acknowledge mode
- Destination type

Bindings properties

Specify bindings assembly properties for the message bean.

- Listener port name

EJB name

The logical name for the message bean (as an enterprise bean)

The logical name for the message bean (as an enterprise bean). This name must be unique within the EJB module. There is no relationship between this name and the JNDI name.

Data type String

Display name

A short name that is intended to be displayed by graphical user interfaces

Data type String

Description

A description of the message bean, for administrative use

Data type String

EJB class

The full package name of the message bean class

Specify the full package name of the message bean class, for example, `com.ibm.ejs.doc.account.MessageBean`. You can either type the class name or click **Browse** to locate an existing class file.

Data type String

Transaction type

Whether the message bean manages its own transactions or the container manages transactions on behalf of the bean

Whether the message bean manages its own transactions or the container manages transactions on behalf of the bean. All messages retrieved from a specific destination have the same transactional behavior. To enable the transactional behavior that you want, you must configure the JMS destination with the same transactional behavior as you configure for the message bean.

Data type	Enum
Default	Bean
Range	Bean The message bean manages its own transactions Container The container manages transactions on behalf of the bean

Message selector

The JMS message selector to be used to determine which messages the message bean receives

The JMS message selector to be used to determine which messages the message bean receives; for example:

```
JMSType='car' AND color='blue' AND weight>2500
```

The selector string can refer to fields in the JMS message header and fields in the message properties. Message selectors cannot reference message body values.

Data type	String
Range	A String whose syntax is based on a subset of the SQL92 conditional syntax.

Acknowledge mode

How the session acknowledges any messages it receives.

This property applies only to message-driven beans that uses bean-managed transaction demarcation (**Transaction type** is set to Bean).

Data type	Enum
Default	Auto Acknowledge
Range	Auto Acknowledge The session automatically acknowledges a message when it has either successfully returned from a call to receive, or the message listener it has called to process the message successfully returns. Dups OK Acknowledge The session lazily acknowledges the delivery of messages. This is likely to result in the delivery of some duplicate messages if JMS fails, so it should be used only by consumers that are tolerant of duplicate messages.

Destination type

Whether the message bean uses a queue or topic destination.

Data type	Enum
Default	Null
Range	Queue The message bean uses a queue destination. Topic The message bean uses a topic destination.

Listener port name

The name of the listener port for this message bean.

The name of the listener port for this message bean (as defined on the WebSphere administrative console).

Data type	String
------------------	--------

Method extensions

Method extensions are IBM extensions to the standard deployment descriptors for enterprise beans.

Method extension properties are used to define transaction isolation levels for methods, to control the delegation of a principal's credentials, and to define custom finder methods.

Method extension assembly settings

Method extensions are IBM extensions to the standard J2EE deployment descriptors for Enterprise JavaBeans (EJB) Version 1.x-compliant beans. Method extension settings define transaction isolation levels for methods and control the delegation of a principal's credentials.

Method type

Specifies the type of the enterprise bean method.

Data type	String
Range	Valid values are Home, Remote, and Unspecified.

Name

Specifies the name of an enterprise bean method, or the asterisk character (*). The asterisk is used to denote all methods of an enterprise bean's remote and home interfaces.

Parameters

Contains a list of fully qualified Java type names of the method parameters. Used to identify a single method among multiple methods with an overloaded method name.

Isolation level attributes

The transaction isolation level determines how isolated one transaction is from another. This can be set for individual methods in an enterprise bean or for all methods in the enterprise bean. An asterisk is used to indicate all methods in the bean. This setting is not applicable for EJB 2.x-compliant beans.

Within a transactional context, the isolation level associated with the first method call becomes the required isolation level for all methods called within that

transaction. If a method is called with a different isolation level from that of the first method, the java.rmi.RemoteException exception is thrown.

Isolation level

Specifies the level of transactional isolation.

The container uses the transaction isolation level attribute as follows:

- Session beans and entity beans with bean-managed persistence (BMP): For each database connection used by the bean, the container sets the transaction isolation level at the start of each transaction unless the bean explicitly sets the isolation level on the connection.
- Entity beans with container-managed persistence (CMP): The container generates database access code that implements the specified isolation level.

Data type

String

Range

Valid values are Serializable, Repeatable read, Read committed, and Read uncommitted

Serializable

This level prohibits the following types of reads:

- *Dirty* reads, in which a transaction reads a database row containing uncommitted changes from a second transaction.
- *Nonrepeatable* reads, in which one transaction reads a row, a second transaction changes the same row, and the first transaction rereads the row and gets a different value.
- *Phantom* reads, in which one transaction reads all rows that satisfy an SQL WHERE condition, a second transaction inserts a row that also satisfies the WHERE condition, and the first transaction applies the same WHERE condition and gets the row inserted by the second transaction.

Repeatable read

This level prohibits dirty reads and nonrepeatable reads, but it allows phantom reads.

Read committed

This level prohibits dirty reads but allows nonrepeatable reads and phantom reads.

Read uncommitted

This level allows dirty reads, nonrepeatable reads, and phantom reads.

Access intent - Intent type

Specifies whether to load the enterprise bean as read-only or for update. This setting is applicable only for EJB 1.x-compliant beans.

This setting is applicable for the following types of beans:

- EJB 1.x-compliant entity beans
- Enterprise beans with CMP version 1.x that are packaged in EJB 2.x-compliant modules

To specify the access intent for EJB 2.x-compliant beans, select an access intent policy.

Data type

String

Range

Valid values are Read or Update

Finder descriptor - User

Specifies that the user has provided a finder helper class in the entity bean's home interface. The class contains specialized finder methods. This setting is applicable only for EJB 1.x-compliant entity beans.

Finder descriptor - EJB QL

Describes the semantics of a finder method that uses EJB QL (Enterprise JavaBeans query language). This setting is applicable only for EJB 1.x-compliant entity beans. EJB QL is a declarative, SQL-like language that is intended to be compiled to the target language of the persistent datastore used by a persistence manager. The language is independent of the bean's mapping to a relational datastore and is therefore portable. The EJB query specifies a search based on the persistent attributes and relationships of the bean. An EJB query can contain the following clauses:

- SELECT (optional), which specifies the EJB objects to return
- FROM (required), which specifies the collections of objects to which the query is to be applied
- WHERE (optional), which contains search predicates over the collections
- ORDER BY (optional), which specifies the ordering of the resulting collection

Finder descriptor - Full SELECT

Describes the semantics of a finder method that uses an SQL SELECT clause. For information on restrictions, see the documentation for the Deployment Tool for Enterprise JavaBeans.

Finder descriptor - WHERE clause

Describes the semantics of a finder method that uses an SQL WHERE clause. This clause restricts the results that are returned by the query. For information on restrictions, see the documentation for the Deployment Tool for Enterprise JavaBeans.

Security identity

Specifies whether a principal's credential settings are to be handled as indicated in the **Run-As mode** setting. If this is enabled, the **Run-As mode** setting can be edited.

Description

Contains further information about the security instructions.

Run-As mode

Specifies the credential information to be used by the security service to determine the permissions that a principal has on various resources.

At appropriate points, the security service determines whether the principal is authorized to use a particular resource based on the principal's permissions. If the method call is authorized, the security service acts on the principal's credential settings according to the **Run-As mode** setting of the enterprise bean.

Data type

Enumerated integer

Range

Valid values are Use identity of caller, Use identity of EJB server, and Use identity assigned to specified role

Additional information about valid values for this setting follows:

Use identity of caller

The security service makes no changes to the principal's credential settings.

Use identity of EJB server

The security service alters the principal's credential settings to match the credential settings associated with the EJB server.

Use identity assigned to specified role

A principal that has been assigned to the specified security role is used for the execution of the bean's methods. This association is part of the

application binding in which the role is associated with a user ID and password of a user who is granted that role.

Role name

Specifies the name of a security role. If **Run-As mode** is set to Use identity assigned to specified role, a principal that has been granted this role is used.

Description

Contains further information about the security role.

Method permissions

A method permission is a mapping between one or more security roles and one or more methods that a member of the role can call.

Method permission assembly settings

A method permission is a mapping between one or more security roles and one or more methods that a member of the role can call. Assembly settings for method permissions include an optional description, a list of security role names, and a list of methods. The security roles must be defined, and the methods must be defined in the enterprise bean's remote or home interfaces.

Method permission name

Specifies a name for the mapping between method permissions and security roles.

Description

Contains text that describes the mapping between method permissions and security roles.

Methods - Name

Specifies the name of an enterprise bean method, or the asterisk (*) character. The asterisk is used to denote all the methods of an enterprise bean's remote and home interfaces.

Methods - Enterprise bean

Specifies the name of the enterprise bean that contains the method.

Methods - Type

Distinguishes between a method with the same signature that is defined in both the home and remote interface. Use Unspecified if a method permission applies to all methods of a bean.

Data type

String

Range

Valid values are Unspecified, Remote, or Home.

Methods - Parameters

Contains a list of fully qualified Java type names of the method parameters. This setting is used to identify a single method among multiple methods with an overloaded method name.

Unchecked

Specifies whether the method permission is checked before the method is run.

Roles - Role name

Specifies the name of the security role that must be granted in order to call the method.

Query assembly settings

Use these to specify a finder or SELECT query.

A query element contains the following:

- Optional description of the query
- Name of the finder or SELECT method that uses the query
- The return type of mapping, if it is used
- Whether the query is for a SELECT method
- EJB query language (EJB QL) query string that defines the query

Queries that are expressed in EJB QL must use the `ejb-ql` element to specify the query. If a query cannot be expressed in EJB QL, describe the semantics of the query by using the description element and leave the `ejb-ql` element empty.

Name

Contains the name of an enterprise bean method or the asterisk (*) character. An asterisk in the method-name element denotes all methods of an enterprise bean's remote and home interfaces.

Parameters

Contains a list of the fully-qualified Java names of the method parameters.

Result type

Used in the query element to indicate whether a returned abstract schema type for a SELECT method should be mapped to an `EJBLocalObject` or `EJBObject` type.

References

References are logical names used to locate external resources for enterprise applications. References are defined in the application's deployment descriptor file. At deployment, the references are bound to the physical location (global JNDI name) of the resource in the target operational environment.

This product supports the following types of references:

- An EJB reference is a logical name used to locate the home interface of an enterprise bean.
- A resource reference is a logical name used to locate a connection factory object.

These objects define connections to external resources such as databases and messaging systems. The container makes references available in a JNDI naming subcontext. By convention, references are organized as follows:

- EJB references are made available in the `java:comp/env/ejb` subcontext.
- Resource references are made available as follows:
 - JDBC `DataSource` references are declared in the `java:comp/env/jdbc` subcontext.
 - JMS connection factories are declared in the `java:comp/env/jms` subcontext.
 - JavaMail connection factories are declared in the `java:comp/env/mail` subcontext.
 - URL connection factories are declared in the `java:comp/env/url` subcontext.

EJB reference assembly settings

An EJB reference is a logical name used to locate the home interface of an enterprise bean used by an application.

At deployment, the EJB reference is bound to the enterprise bean's home in the target operational environment. The container makes the application's EJB references available in a JNDI naming context. It is recommended that references to enterprise beans be organized in the `ejb` subcontext of the application's environment (in `java:comp/env/ejb`).

Name

Specifies the JNDI name of the enterprise bean's home interface relative to the `java:comp/env` context.

For example, if `ejb/EmplRecord` is specified, the referring code looks up the enterprise bean's home interface at `java:comp/env/ejb/EmplRecord`. This JNDI name is an alias used by the code (the actual JNDI name is specified on the Binding tab).

Description

Contains text that describes the EJB reference.

Link

Used to link an EJB reference to an enterprise bean in the current module (the same module as the one making the reference) or in another module within the same J2EE application. This setting specifies the name of the target enterprise bean. The target enterprise bean can be in any EJB module in the same J2EE application as the referring module. To avoid having to rename enterprise beans to have unique names within an J2EE application, specify the path name of the EJB archive file that contains the referenced enterprise bean and append the target bean's name, separated by a # symbol (for example, `...products/product.jar#ProductEJB`). The path name is relative to the referring module's archive file specification. If a link is not specified, the reference must be resolved to a JNDI name during installation.

Home

Specifies the fully qualified name of the enterprise bean's home interface (for example, `com.ibm.ejbs.EmplRecordHome`).

Remote

Specifies the fully qualified name of the enterprise bean's remote interface (for example, `com.ibm.ejbs.EmplRecord`).

Type

Specifies the expected type of the referenced enterprise bean.

Data type	String
Default	None; must be set
Range	Entity or Session

JNDI name

Binding information that is used by the run-time environment to resolve the location of a resource.

For EJB references, the value of this setting must match the JNDI name of the enterprise bean as it was specified on the **Binding** tab for the EJB module that contains the bean.

EJB local-reference assembly settings

For EJB 2.0-compliant beans, the EJB local reference element declares a reference to another enterprise bean's local home interface.

Name

Specifies the name of an EJB reference.

This is the JNDI name that the servlet code uses to get a reference to the enterprise bean. The following example illustrates how this element is specified in the deployment descriptor:

```
<ejb-ref-name>ejb/Payroll</ejb-ref-name>
```


Description

Contains a description of the parent element. This can include any information that the EJB archive-file producer wants to provide to the consumer of the EJB archive file.

Link

Used in the `ejb-ref` element to specify that an EJB reference is linked to an enterprise bean in the encompassing web-application package. The value of the link element must be the EJB name of an enterprise bean in the same web-application package. The following example illustrates how this element is specified in the deployment descriptor:

```
<ejb-link>EmployeeRecord</ejb-link>
```

Local interface

Specifies the fully-qualified name of the enterprise bean's local interface.

Local home

Specifies the fully-qualified name of the enterprise bean's local home interface.

Type

Specifies the expected type of the referenced enterprise bean.

EJB relation assembly settings

An EJB relation describes a relationship between two entity beans with container-managed persistence.

The name of the relationship, if specified, is unique within an EJB archive file.

Description

Contains text to describe the EJB relationship role.

Source EJB

Specifies the source of the role that participates in a relationship.

Multiplicity

Specifies the multiplicity of the role that participates in a relation.

Cascade delete

Within a particular relationship, specifies that the lifetime of one or more entity beans is dependent on the lifetime of another entity bean.

Cascade delete can be specified only for an EJB relationship role contained in an EJB relation in which the other EJB relationship role specifies a multiplicity of one.

CMR field

Enables the declaration of a container-managed relationship (CMR) field.

The CMR field describes the bean provider's view of a relationship. It consists of an optional description and the name and class type of the source enterprise bean's role in a relationship.

Exclude list assembly settings

The exclude list indicates which methods in the enterprise beans may not be called. You should also configure security for the enterprise bean so that access to the listed methods is not permitted.

This capability applies only to Enterprise JavaBeans (EJB) Version 2.x-compliant beans. For more information about exclude lists, see the EJB specification.

Description

Provides additional information about this exclude list.

Methods - Name

Specifies the name of an enterprise bean method, or the asterisk (*) character. The asterisk is used to denote all the methods of an enterprise bean's remote and home interfaces.

Methods - Enterprise bean

Specifies the name of the enterprise bean that contains the method.

Methods - Type

Distinguishes between a method with the same signature that is defined in both the home and remote interface. Use Unspecified if the exclusion applies to all methods of a bean.

Data type

String

Range

Valid values are Unspecified, Remote, or Home

Methods - Parameters

Contains a list of fully qualified Java type names of the method parameters. This setting is used to identify a single method among multiple methods with an overloaded method name.

Security role assembly settings

A security role is a logical grouping of principals. Access to operations (such as enterprise-bean methods) is controlled by granting access to a role.

Role name

Specifies the name of a security role that is unique to an application. This setting applies only when you are specifying security roles at the application level (EAR file).

Description

Contains text that describes the application-specific security role. This setting applies only when you are specifying security roles at the application level (EAR file).

Binding - Groups - Name

Specifies the user groups that are granted the application-specific security role. This setting applies only when you are specifying security roles at the application level (EAR file).

Binding - Users - Name

Specifies the users that are granted the application-specific security role. This setting applies only when you are specifying security roles at the application level (EAR file).

Binding - Special Subjects - Name

Specifies one of two special categories of authenticate users to which application-specific security roles can be granted: Everyone or All. This setting applies only when you are specifying security roles at the application level (EAR file).

If the special subject All is granted a role, any user who can authenticate by using a valid user ID and password is considered to be granted that role.

If the special subject Everyone is granted a role, all users, including those who did not authenticate, are granted the role. In other words, a method on an enterprise bean or a URI is unprotected if any of the required roles for that method are granted to the special subject Everyone.

Data type	String
Range	Valid values are All or Everyone

Session bean assembly properties

A session bean encapsulates transient data that is associated with a particular EJB client. Unlike data in an entity bean, the data in a session bean is not stored in a persistent data source.

EJB name

Specifies a logical name for the enterprise bean. This name must be unique within the EJB module. There is no relationship between this name and the JNDI name.

Display name

Specifies a short name that is intended to be displayed by GUIs.

Description

Contains text that describes the session bean.

EJB class

Specifies the full name of the enterprise bean class (for example, `com.ibm.ejs.doc.account.AccountBean`).

Remote - Home

Specifies the full name of the enterprise bean's home interface class (for example, `com.ibm.ejs.doc.account.AccountHome`).

Remote - Interface

Specifies the full name of the enterprise bean's remote interface class (for example, `com.ibm.ejs.doc.account.Account`).

Local interface - Home

Specifies the full name of the enterprise bean's home interface class (for example, `com.ibm.ejs.doc.account.AccountLocalHome`).

Local interface - Interface

Specifies the full name of the enterprise bean's local interface class (for example, `com.ibm.ejs.doc.account.AccountLocal`).

Session type

Specifies whether the enterprise bean maintains a conversational state (is stateful) or does not (is stateless).

Data type	String
Range	Valid values are Stateful and Stateless

Transaction type

Specifies whether the enterprise bean manages its own transactions or whether the container manages transactions on behalf of the bean.

Data type	String
Range	Valid values are Container or Bean

Small icon

Specifies the name of a JPEG or GIF file that contains a small image (16x16 pixels). The image is used as an icon to represent the session bean in a GUI.

Large icon

Specifies the name of a JPEG or GIF file that contains a large image (32x32 pixels). The image is used as an icon to represent the session bean in a GUI.

Security identity

Specifies whether a principal's credential properties are to be handled as indicated in the **Run-As mode** property. If this setting is enabled (that is, set to true), the **Run-As mode** setting can be edited.

Description

Contains further information about the security instructions.

Run-As mode

Specifies the credential information to be used by the security service to determine the permissions that a principal has on various resources.

At appropriate points, the security service determines whether the principal is authorized to use a particular resource based on the principal's permissions. If the method call is authorized, the security service acts on the principal's credential properties according to the **Run-As mode** setting of the enterprise bean.

Data type

Enumerated integer

Range

Valid values are Use identity of caller and Use identity assigned to specified role

Additional information about valid values for this setting follows:

Use identity of caller

The security service makes no changes to the principal's credential properties.

Use identity assigned to specified role

A principal that has been assigned to the specified security role is used for the execution of the bean's methods. This association is part of the application binding in which the role is associated with a user ID and password of a user who is granted that role.

Role name

Specifies the name of a security role. If **Run-As mode** is set to Use identity assigned to specified role, a principal that has been granted this role is used.

Description

Contains further information about the security role.

Timeout

This property applies only to stateful session beans.

This property is an IBM extension to the standard J2EE deployment descriptor.

Data type

Integer

Units

Seconds

Inheritance root

Specifies whether the enterprise bean is at the root of an inheritance hierarchy.

This property is an IBM extension to the standard J2EE deployment descriptor.

Bean Cache - Activate at

Specifies the point at which an enterprise bean is activated and placed in the cache. Removal from the cache and passivation is also governed by this setting. This setting applies to stateful session beans only (not to stateless beans).

This property is an IBM extension to the standard J2EE deployment descriptor.

Data type	String
Default	Once
Range	Valid values are Once and Transaction

Additional information about valid values follows:

Once Indicates that the bean is activated when it is first accessed in the server process, and passivated (and removed from the cache) at the discretion of the container, for example, when the cache becomes full.

Transaction

Indicates that the bean is activated at the start of a transaction and passivated (and removed from the cache) at the end of the transaction.

Local Transactions - Unresolved action

Specifies the action that the EJB container must take if resources are uncommitted by an application in a local transaction.

This property is an IBM extension to the standard J2EE deployment descriptor. This setting is applicable only when **Resolution control** is set to Application. A local transaction context is created when a method runs in what the EJB specification refers to as an unspecified transaction context.

Data type	String
Default	Rollback
Range	Valid values are Commit and Rollback

Additional information about these settings follows:

Commit

At end of the local transaction context, the container instructs all unresolved local transactions to commit.

Rollback

(Default) At end of the local transaction context, the container instructs all unresolved local transactions to roll back.

Local Transactions - Resolution control

Specifies how the local transaction is to be resolved before the local transaction context ends: by the application through user code or by the EJB container.

This property is an IBM extension to the standard J2EE deployment descriptor.

Data type	String
Range	Valid values are Application and ContainerAtBoundary

Additional information about these settings follows:

Application

When this setting is used, your code must either commit or roll back the local transaction. If this does not occur, the runtime environment logs a warning and automatically commits or rolls back the connection as specified by the **Unresolved action** setting.

ContainerAtBoundary

When this setting is used, the container takes responsibility for resolving each local transaction. This provides you with a programming model similar to global transactions in which your code simply gets a connection and performs work within it. User code does not have to handle local transactions.

- If the **Boundary** attribute is set to `ActivitySession`, then the local transactions are enlisted as `ActivitySession` resources and directed to complete by the `ActivitySession`.
- If the the **Boundary** attribute is set to `BeanMethod`, then the local transactions are committed at method end by the container.

Connections are never committed automatically by the resource adapter when this value is configured for the bean **Unresolved action** is not used. An application cannot call `Connection.LocalTransaction.begin()` when using this policy and receives an exception from the resource adapter if it does so.

When using a **Resolution control** of `ContainerAtBoundary`, applications must get connection handles *after* the local transaction context boundary has been started by the container. The application should close the connection before the end of the boundary, although any work performed on the connection is not committed or rolled back until the local transaction context ends. This model of connection usage is sometimes referred to as the “get-use-close” model.

This value is supported only for EJB components that use container-managed transactions. It is not supported for web components or for enterprise beans that use bean-managed transactions.

Local Transactions - Boundary

Specifies the duration of a local transaction context. This property does not apply to stateless session beans.

This property is an IBM extension to the standard J2EE deployment descriptor.

Data type	String
Default	BeanMethod
Range	Valid values are BeanMethod and ActivitySession

Additional information about valid settings follows:

BeanMethod

When this setting is used, the local transaction begins when the method begins and ends when the method ends.

ActivitySession

When this setting is used, the local transaction must be resolved within the scope of any `ActivitySession` in which it was started or, if no `ActivitySession` context is present, within the same bean method in which it was started.

This property can be changed on WAS Enterprise only.

JNDI name

Specifies the JNDI name of the bean’s home interface. This is the name under which the enterprise bean’s home interface is registered and therefore, is the name that must be specified when an EJB client does a lookup of the home interface.

EJB containers

An Enterprise JavaBeans (EJB) container provides a run-time environment for enterprise beans within the application server. The container handles all aspects of an enterprise bean's operation within the application server and acts as an intermediary between the user-written business logic within the bean and the rest of the application server environment.

One or more EJB modules, each containing one or more enterprise beans, can be installed in a single container.

The EJB container provides many services to the enterprise bean, including the following:

- Beginning, committing, and rolling back transactions as necessary.
- Maintaining pools of enterprise bean instances ready for incoming requests and moving these instances between the inactive pools and an active state, ensuring that threading conditions within the bean are satisfied.
- Most importantly, automatically synchronizing data in an entity bean's instance variables with corresponding data items stored in persistent storage.

By dynamically maintaining a set of active bean instances and synchronizing bean state with persistent storage when beans are moved into and out of active state, the container makes it possible for an application to manage many more bean instances than could otherwise simultaneously be held in the application server's memory. In this respect, an EJB container provides services similar to virtual memory within an operating system.

Between transactions, the state of an entity bean can be cached. The EJB container supports option A, B, and C caching.

- With option A caching, the application server assumes that the entity bean is used within a single container. Clients of that bean must direct their requests to the bean instance within that container. The entity bean has exclusive access to the underlying database, which means that the bean cannot be cloned or participate in workload management if option A caching is used.
- With option B caching, the entity bean remains active in the cache throughout the transaction but is reloaded at the start of each method call.
- With option C caching (the default), the entity bean is always reloaded from the database at the beginning of each transaction. A client can attempt to access the bean and start a new transaction on any container that has been configured to host that bean. This is similar to the session clustering facility described for HTTP sessions in that the entity bean's state is maintained in a shared database that can be accessed from any server when required.

This product supports the cloning of stateful session bean home objects among multiple application servers. However, it does not support the cloning of a specific instance of a stateful session bean. Each instance of a particular stateful session bean can exist in just one application server and can be accessed only by directing requests to that particular application server. State information for a stateful session bean cannot be maintained across multiple members of a server cluster.

For more information about EJB containers, see "Resources for learning."

Managing EJB containers

Each application server can have a single EJB container; one is created automatically for you when the application server is created. The following steps are to be performed only as needed to improve performance after the EJB application has been deployed.

1. Adjust EJB container settings.
2. Adjust EJB cache settings.

If adjustments do not improve performance, consider adjusting access intent policies for entity beans, reassembling the module, and redeploying the module in the application.

EJB container settings

Use this page to configure and manage a specific EJB container.

To view this administrative console page, click **Servers > Application Servers > *serverName* > EJB Container**.

Passivation directory

Specifies the directory into which the container saves the persistent state of passivated stateful session beans.

Beans are passivated when the number of active bean instances becomes greater than the cache size specified in the container configuration. When a stateful bean is passivated, the container serializes the bean instance to a file in the passivation directory and discards the instance from the bean cache. If, at a later time, a request arrives for the passivated bean instance, the container retrieves it from the passivation directory, deserializes it, returns it to the cache, and dispatches the request to it. If any step fails (for example, if the bean instance is no longer in the passivation directory), the method invocation fails.

Inactive pool cleanup interval

Specifies 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.

Data type	Integer
Units	Milliseconds
Range	Greater than 0

Default datasource JNDI name

Specifies the JNDI name of a data source to use if no data source is specified during application deployment. This setting is not applicable for EJB 2.x-compliant CMP beans.

Servlets and enterprise beans use *data sources* to obtain these connections. When configuring a container, you can specify a default data source for the container. This data source becomes the default data source used by any entity beans installed in the container that use container-managed persistence (CMP).

The default data source for a container is secure. When specifying it, you must provide a user ID and password for accessing the data source.

Specifying a default data source is optional if each CMP entity bean in the container has a data source specified in its configuration. If a default data source is

not specified and a CMP entity bean is installed in the container without specifying a data source for that bean, applications cannot use that CMP entity bean.

Initial state

Specifies the execution state requested when the server first starts.

Data type	String
Default	Started
Range	Valid values are Started and Stopped

EJB container system properties

In addition to the settings accessible from the administrative console, you can set the following system property by command-line scripting:

com.ibm.websphere.ejbcontainer.poolSize

Specifies the size of the pool for the specified bean type. This property applies to stateless, message-driven and entity beans. If you do not specify a default value, the container defaults of 50 and 500 are used.

Set the pool size for a given entity bean as follows:

```
beantype=min,max[:beantype=min,max...]
```

beantype is the J2EE name of the bean, formed by concatenating the application name, the # character, the module name, the # character, and the name of the bean (that is, the string assigned to the <ejb-name> field in the bean's deployment descriptor). *min* and *max* are the minimum and maximum pool sizes, respectively, for that bean type. Do not specify the square brackets shown in the previous prototype; they denote optional additional bean types that you can specify after the first. Each bean-type specification is delimited by a colon (:).

Use an asterisk (*) as the value of *beantype* to indicate that all bean types are to use those values unless overridden by an exact bean-type specification somewhere else in the string, as follows:

```
*=30,100
```

To specify that a default value be used, omit either *min* or *max* but retain the comma (,) between the two values, as follows (split for publication):

```
SMAApp#PerfModule#TunerBean=54,  
:SMAApp#SModule#TypeBean=100,200
```

You can specify the bean types in any order within the string.

EJB cache settings

Use this page to configure and manage the cache for a specific EJB container. To determine the cache absolute limit, multiply the number of enterprise beans active in any given transaction by the total number of concurrent transactions expected. Then, add the number of active session bean instances.

To view this administrative console page, click **Servers > Application Servers > serverName > EJB Container > EJB Cache Settings**.

Cleanup interval

Specifies the interval at which the container attempts to remove unused items from the cache in order to reduce the total number of items to the value of the cache size.

The cache manager tries to maintain some unallocated entries that can be allocated quickly as needed. A background thread attempts to free some entries while maintaining some unallocated entries. If the thread runs while the application server is idle, when the application server needs to allocate new cache entries, it does not pay the performance cost of removing entries from the cache. In general, increase this parameter as the cache size increases.

Data type	Integer
Units	Milliseconds
Range	Greater than 0
Default	3000

Cache size

Specifies the number of buckets in the active instance list within the EJB container. A bucket can contain more than one active enterprise bean instance, but performance is maximized if each bucket in the table has a minimum number of instances assigned to it. When the number of active instances within the container exceeds the number of buckets, that is, the cache size, the container periodically attempts to reduce the number of active instances in the table by passivating some of the active instances. For the best balance of performance and memory, set this value to the maximum number of active instances expected during a typical workload.

Data type	Integer
Units	Buckets in the hash table
Range	Greater than 0. The container selects the next largest prime number equal to or greater than the specified value.
Default	2053

Container interoperability

Container interoperability describes the ability of WebSphere Application Server clients and servers at different versions to successfully negotiate differences in native Enterprise JavaBeans (EJB) Version 1.1 finder methods support and Java 2 Platform, Enterprise Edition (J2EE) Version 1.3 compliance.

At one time, there were significant interoperability problems among WebSphere Application Server, versions 4.0.x and 3.5.x distributed, and Version 4.0.x for zSeries. The introduction of interoperable versions of some class types solved these problems for distributed versions 3.5.6, 4.0.3, and 5 as well as for zSeries Version 4.0.x.

Older 4.0.x and 3.5.x client and application server versions do not support the interoperability classes, which makes them uninteroperable with versions that use the classes. The system property `com.ibm.websphere.container.portable` remedies this situation by enabling newer versions of the application server to turn off the interoperability classes. This lets a more recent application server return class types that are interoperable with an older client.

Depending on the value of `com.ibm.websphere.container.portable`, application servers at versions 5, 4.0.3 and later, and 3.5.6 and later, return different classes for the following:

- Enumerations and collections returned by EJB 1.1 finder methods
- EJBMetaData
- Handles to:

- Entity beans
- Session beans
- Home interfaces

If the property is set to false, application servers return the old class types, to enable interoperability with versions 3.5.5 and earlier, and 4.0.2 and earlier. If the property is set to true, application servers return the new classes.

Instructions for setting the `com.ibm.websphere.container.portable` property are in the release notes for versions 3.5.6 and later, and 4.0.3 and later. The following tables show interoperability characteristics for various version combinations of application servers and clients as well as default property values for each combination.

Interoperability of Version 3.5.x client with Version 5 application server

Clients at Version 3.5.5 and earlier are not interoperable with Version 5 servers when using:

- EJBMetaData
- Enumerations returned by EJB 1.x finder methods
- Handles to entity beans

If you would like to use updated Handle classes in EJB 2.x-compliant beans but have one of the older clients (versions 3.5.5 and earlier) installed, set the system property `com.ibm.websphere.container.portable.finder` to false. With this setting in place, the Version 5 application server uses the updated handles but returns the enumerations and collections that were used in the earlier clients.

To interoperate with Version 5 application servers, you must upgrade all Version 3.5.x clients to Version 3.5.6 or later.

Interoperability of Version 5 client with Version 3.5.x application server

Client at Version 5, using this function	Application server at Version 3.5.6, property true	Application server at Version 3.5.6, property false (default)	Application server at Version 3.5.5 and earlier
EJBMetaData	Does not work across domains	Works	Does not work
Handle to session bean	Works	Works	Does not work
Handle to entity bean	Does not work across domains	Does not work across domains	Does not work across domains
Enumeration returned by EJB 1.x finder method	Works	Works	Works

Interoperability of Version 4.0.x client with Version 5 application server

Ideally, all 4.0.x clients that use Version 5 application servers should be at Version 4.0.3 or later.

Version 5 application servers return the interoperability class types by default (true). This can cause interoperability problems for distributed clients at versions 4.0.1 or 4.0.2. In particular, problems can occur with collections and enumerations returned by EJB 1.1 finder methods.

Although it is strongly discouraged, you can set `com.ibm.websphere.container.portable` to `false` on a Version 5 application server. This causes the application server to return the old class types, providing interoperability with clients at Version 4.0.2 and earlier. This is discouraged because:

- The Version 5 application server instance would become non-J2EE 1.3 compliant with regard to handles, home interface handles, and `EJBMetaData`.
- EJB 1.x finder methods return collection and enumeration objects that do not originate from `ejbportable.jar`.
- Interoperability restrictions still exist with the property set to `false`.
- Version 5 client handles to entity beans and home interfaces do not work across domains for the server you set to `false`.

If you would like to use updated Handle classes in EJB 2.x-compliant beans but have one of the older clients (versions 4.0.2 and earlier) installed, set the system property `com.ibm.websphere.container.portable.finder` to `false`. With this setting in place, the Version 5 application server uses the updated handles but returns the enumerations and collections that were used in the earlier clients.

Interoperability of client at Version 4.0.2 and earlier with Version 5 application server

Client at Version 4.0.2 and earlier, using this function	Application server at Version 5, property true (default)	Application server at Version 5, property false
<code>EJBMetaData</code>	Does not work	Works for 4.0.2 client
Handle to session bean	Does not work	Works
Handle to entity bean	Does not work	Does not work across cells
Enumeration returned by EJB 1.x finder method	Does not work	Works
Collection returned by EJB 1.x finder method	Does not work	Works
Handle to home interface	Does not work	Does not work across cells

If you would like to use updated Handle classes in EJB 2.x-compliant beans but have one of the older clients (versions 3.5.5 and earlier, and 4.0.2 and earlier) installed, set the system property `com.ibm.websphere.container.portable.finder` to `false`. With this setting in place, the Version 5 server uses the new Handle classes but returns the older enumeration and collection classes.

Interoperability of client at Version 4.0.3 and later with Version 5 application server

Clients at Version 4.0.3 and later work well with Version 5 application servers. However, if you set the `com.ibm.websphere.container.portable` to `false`, client handles to entity beans and home interfaces do not work across domains for the server you set to `false`.

Client at Version 4.0.3 and later, using this function	Application server at Version 5, property true (default)	Application server at Version 5, property false
EJBMetaData	Works	Works
Handle to session bean	Works	Works
Handle to entity bean	Works	Does not work across cells
Enumeration returned by EJB 1.x finder method	Works	Works
Collection returned by EJB 1.x finder method	Works	Works
Handle to home interface	Works	Does not work across cells

Interoperability of Version 5 client with Version 4.0.x application server

Clients at Version 5 work well with Version 4.0.3 application servers if you set `com.ibm.websphere.container.portable` to `true`. Client handles to entity beans and home interfaces do not work across domains for any Version 4.0.3 server with `com.ibm.websphere.container.portable` at the default value, `false`. Version 5 client handles to application servers at Version 4.0.2 and earlier also have restrictions.

Client at Version 5, using this function	Application server at Version 4.0.3, property true	Application server at Version 4.0.3, property false (default)	Application server at Version 4.0.2 or earlier
EJBMetaData	Works	Works	Works for 4.0.2 server only
Handle to session bean	Works	Works	Works
Handle to entity bean	Works	Does not work across domains	Does not work across domains
Enumeration returned by EJB 1.x finder method	Works	Works	Works
Collection returned by EJB 1.x finder method	Works	Works	Works
Handle to home interface	Works	Does not work across domains	Does not work across domains

Interoperability of zSeries Version 4.0.x client with Version 5 application server

The only valid configuration for container interoperability with zSeries Version 4.0.x clients is the default configuration for the Version 5 application server.

Interoperability of Version 5 client with zSeries Version 4.0.x application server

Version 5 clients should work with a zSeries Version 4.0.x application server with the correct interoperability fixes described in the zSeries documentation. The interoperability characteristics should be the same as for a Version 4.0.3 distributed application server with the property set to `true`.

Client at Version 5, using this function	zSeries application server at Version 4.0.x
EJBMetaData	Works
Handle to session bean	Works
Handle to entity bean	Works
Enumeration returned by EJB 1.x finder method	Works
Collection returned by EJB 1.x finder method	Works
Handle to home interface	Works

Deploying EJB modules

Assemble one or more EJB modules, assemble one or more Web modules, and assemble them into a J2EE application.

1. Prepare the deployment environment.
2. Deploy the application.
3. Update the configuration for each EJB module as needed for the deployment environment.
4. For information about the EJB deployment tool, see the EJB deployment tool.

The next step is to test and debug the module.

EJB module collection

Use this page to manage the EJB modules deployed in a specific application.

To view this administrative console page, click **Applications > Enterprise Applications > *applicationName* > EJB modules**. Click the check boxes to select one or more of the EJB modules in your collection.

URI

When resolved relative to the application URL, this specifies the location of the module's archive contents on a file system. The URI matches the <ejb> or <web> tag in the <module> tag of the application deployment descriptor.

EJB module settings

Use this page to configure and manage a specific deployed EJB module.

Note: You cannot start or stop an individual EJB module for modification. You must start or stop the appropriate application entirely.

To view this administrative console page, click **Applications > Enterprise Applications > *applicationName* > EJB modules > *moduleName***.

URI

When resolved relative to the application URL, this specifies the location of the module archive contents on a file system. The URI must match the URI of a ModuleRef URI in the deployment descriptor of the deployed application (EAR).

Alternate DD

Specifies a deployment descriptor to be used at run time instead of the one installed in the module.

Starting weight

Specifies the order in which modules are started when the server starts. The module with the lowest starting weight is started first.

Data type	Integer
Default	5000
Range	Greater than 0

Troubleshooting tips for EJBDEPLOY relationships

Problems may exist when EJBDeploy creates a data relationship in DB2 for z/OS Version 7.x. EJBDeploy creates a table with a composite of the two primary keys of the EJBs that are related to each other. If the composite keys are larger than 254 characters, DB2 for z/OS V7.x will not accept this relationship and the user will be confronted with errors such as:

```
DSNT408I SQLCODE = -613, ERROR: THE PRIMARY KEY OR A UNIQUE CONSTRAINT
IS TOO LONG OR HAS TOO MANY COLUMNS
DSNT418I SQLSTATE = 54008 SQLSTATE RETURN CODE
```

This problem can be seen when the primary keys that are created for the two related beans have primary keys that are strings. This results in the composite being made up of 2 varchar(250) primary keys for a total of 500, which is greater than 254 maximum in DB2 for z/OS version 7.x.

Things to consider when utilizing top-down mappings to ensure you do not experience this problem:

- Top-down mappings are a guideline and must be reviewed with the DBA.
- Schemas created 'top-down' by EJBDeploy are designed only for testing, and as a guideline for the actual schema required. The use of the 'meet-in-the-middle' mapping does not present this problem.
- The composite key constraint problem is not experienced when using DB2 V8, which has 2K max key lengths.

Enterprise beans: Resources for learning

Use the following links to find relevant supplemental information about enterprise beans. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to this product but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

View links to additional information about:

- "Enterprise beans: Resources for learning"
- "Enterprise beans: Resources for learning"
- "Enterprise beans: Resources for learning"
- "Enterprise beans: Resources for learning"

Planning, business scenarios, and IT architecture

- Mastering Enterprise JavaBeans

A comprehensive treatment of Enterprise JavaBeans (EJB) programming in nonprintable form (PDF). One must be registered to download the PDF, but registration is free. Information about purchasing a hardcopy is available on the Web site.

- *Enterprise JavaBeans* by Richard Monson-Haefel (O'Reilly and Associates, Inc.: Third Edition, 2001)

Programming model and decisions

- Read all about EJB 2.0
 - A comprehensive overview of the specification.
- The J2EE Tutorial
 - This set of articles by Sun Microsystems covers several EJB-related topics, including the basic programming models, persistence, and EJB Query Language.

Programming instructions and examples

- Rules and Patterns for Session Facades
 - EJB programming practice: Fronting entity beans with a session-bean facade.
- WebSphere Application Server Development Best Practices for Performance and Scalability
 - Programming practice for enterprise beans and other types of J2EE components.
- Optimistic Locking in IBM WebSphere Application Server 4.0.2
 - Examples of the effect of optimistic concurrency on application behavior. Although the paper is based on a previous version of this product, the data access issues discussed in it are current.
 - This paper does not seem to be available directly by URL. To view this paper, visit the specified URL and search on "optimistic locking"

Programming specifications

- What's new in the Enterprise JavaBeans 2.0 Specification?
 - You can also download the specification itself from this URL.
- Java™ 2 Platform: Compatibility with Previous Releases
 - This Sun Microsystems article includes both source and binary compatibility issues.

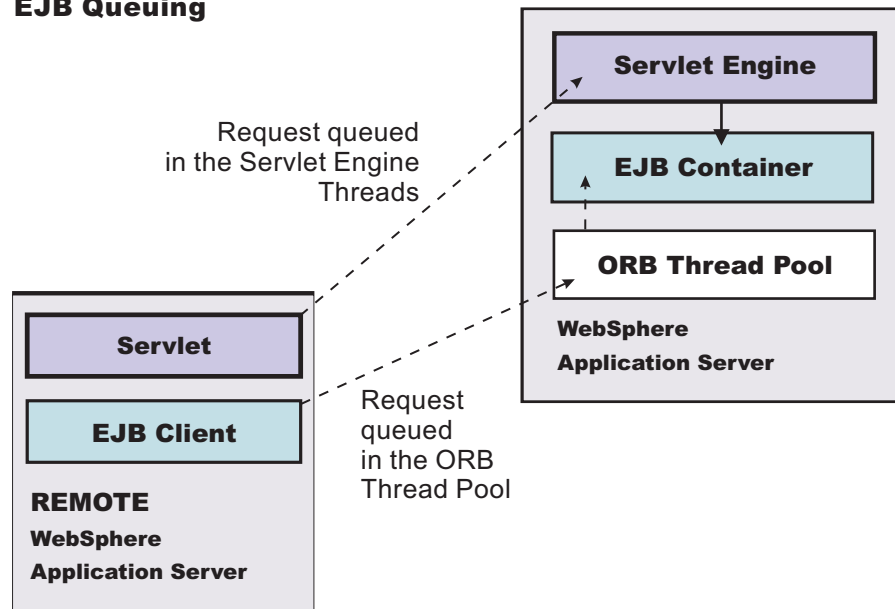
EJB method Invocation Queuing

Method invocations to enterprise beans are only queued for remote clients, making the method call. An example of a remote client is an enterprise Java bean (EJB) client running in a separate Java virtual machine (JVM) (another address space) from the enterprise bean. In contrast, no queuing occurs if the EJB client, either a servlet or another enterprise bean, is installed in the same JVM on which the EJB method runs and on the same thread of execution as the EJB client.

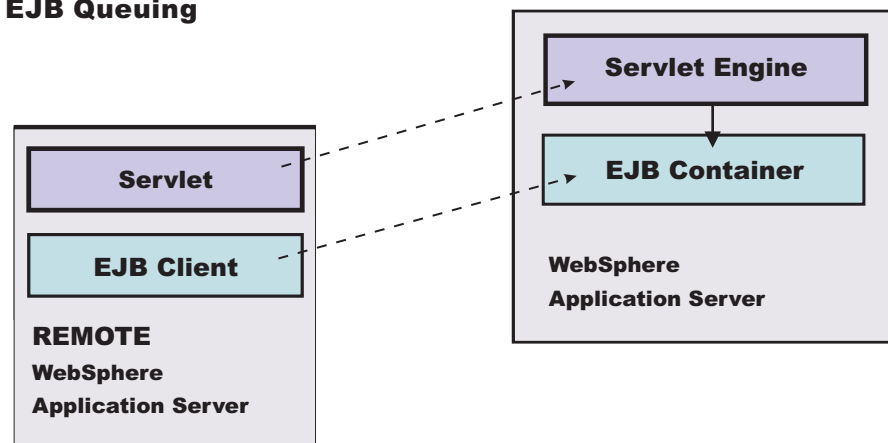
Remote enterprise beans communicate by using the Remote Method Invocation over an Internet Inter-Orb Protocol (RMI-IIOP). Method invocations initiated over RMI-IIOP are processed by a server-side object request broker (ORB). The thread pool acts as a queue for incoming requests. However, if a remote method request is issued and there are no more available threads in the thread pool, a new thread is created. After the method request completes the thread is destroyed. Therefore, when the ORB is used to process remote method requests, the EJB container is an open queue, due to the use of unbounded threads. The following illustration

depicts the two queuing options of enterprise beans.

EJB Queuing



EJB Queuing



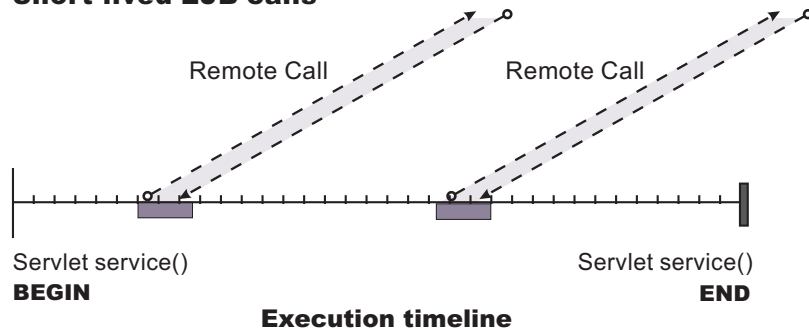
The following are two tips for queueing enterprise beans:

- **Analyze the calling patterns of the EJB client.**

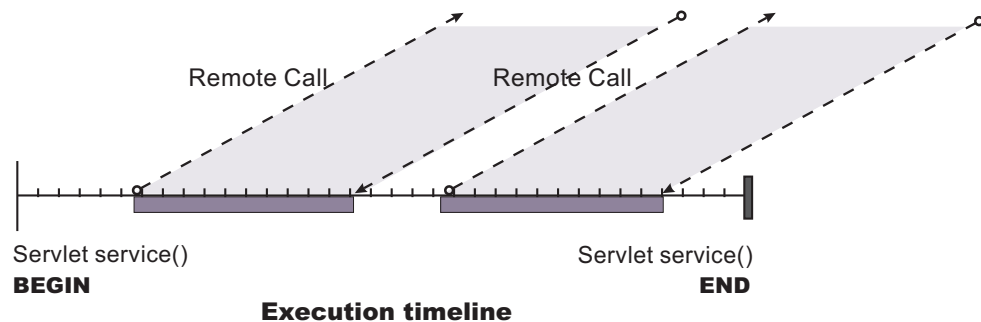
When configuring the thread pool, it is important to understand the calling patterns of the EJB client. If a servlet is making a small number of calls to remote enterprise beans and each method call is relatively quick, consider setting the number of threads in the ORB thread pool to a value lower than the Web

container thread pool size value.

Short-lived EJB calls



Longer-lived EJB calls



The degree to which the ORB thread pool value needs increasing is a function of the number of simultaneous servlets, that is, clients, calling enterprise beans and the duration of each method call. If the method calls are longer or the applications spend a lot of time in the ORB, consider making the ORB thread pool size equal to the Web container size. If the servlet makes only short-lived or quick calls to the ORB, servlets can potentially reuse the same ORB thread. In this case, the ORB thread pool can be small, perhaps even one-half of the thread pool size setting of the Web container.

- **Monitor the percentage of configured threads in use.**

Tivoli Performance Viewer shows a metric called *percent maxed*, which is used to determine how often the configured threads are used. A value that is consistently in the double-digits, indicates a possible bottleneck at the ORB. Increase the number of threads.

See also [Queuing network](#).

Chapter 5. Using message-driven beans in applications

WebSphere Application Server supports asynchronous messaging as a method of communication based on the Java Message Service (JMS) programming interface.

Message-driven beans (a type of enterprise bean defined in the EJB 2.0 specification) extend the base JMS support and the Enterprise JavaBean component model to provide automatic asynchronous messaging. When a message arrives on a destination, a listener passes the message to a new instance of a user-developed message-driven bean for processing.

You can use WebSphere Studio Application Developer to develop applications that use message-driven beans. You can use the WebSphere Application Server runtime tools, like the administrative console, to deploy and administer applications that use message-driven beans.

For more information about implementing WebSphere enterprise applications that use message-driven beans, see the following topics:

- An overview of message-driven beans
- Designing an enterprise application to use a message-driven bean
- Developing an enterprise application to use a message-driven bean
- Deploying an enterprise application to use a message-driven bean
- Configuring message listener resources for message-driven beans
- Troubleshooting problems with message-driven beans

Message-driven beans - an overview

WebSphere Application Server supports automatic asynchronous messaging with message-driven beans (a type of enterprise bean defined in the EJB 2.0 specification). Messaging with message-driven beans is shown in the figure "Message-driven beans - an overview."

The support for message-driven beans is based on the message listener service, which comprises a listener manager that controls and monitors one or more listeners. Each listener monitors a JMS destination for incoming messages. When a message arrives on the destination, the listener passes the message to a new instance of a user-developed message-driven bean (an enterprise bean) for processing. The listener then looks for the next message without waiting for the bean to return.

Messages arriving at a destination being processed by a listener have no client credentials associated with them; the messages are anonymous. Security depends on the role specified by the RunAs Identity for the message-driven bean as an EJB component. For more information about EJB security, see EJB component security.

You are recommended to develop a message-driven bean to delegate the business processing of incoming messages to another enterprise bean, to provide clear separation of message handling and business processing. This also enables the business processing to be invoked by either the arrival of incoming messages or, for example, from a WebSphere J2EE client.

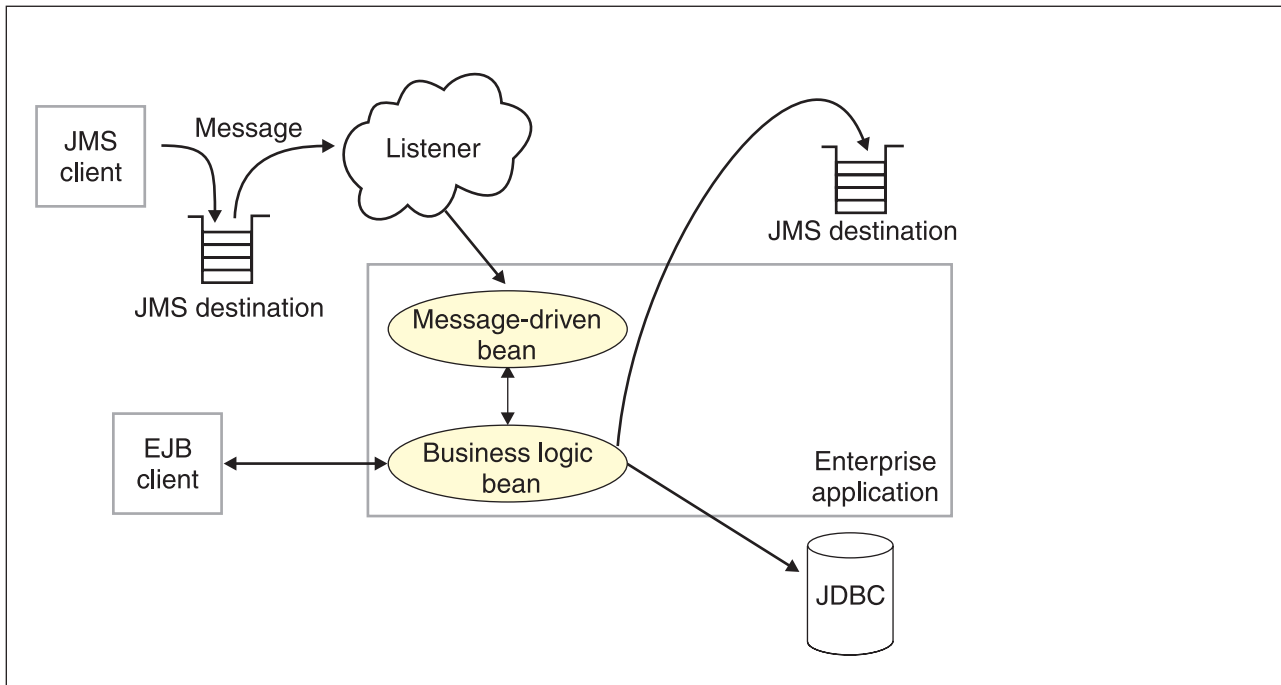


Figure 1. Message-driven beans and the message listener service. This figure shows an incoming message being passed by a JMS listener to a message-driven bean, which passes the message on to a business logic bean for business processing. This messaging is controlled by the listener manager. For more information, see the text that accompanies this figure.

Message-driven beans - components

The WebSphere Application Server support for message-driven beans is based on JMS message listeners and the message listener service, and builds on the base support for JMS. The main components of WebSphere Application Server support for message-driven beans are shown in the following figure and described after the figure:

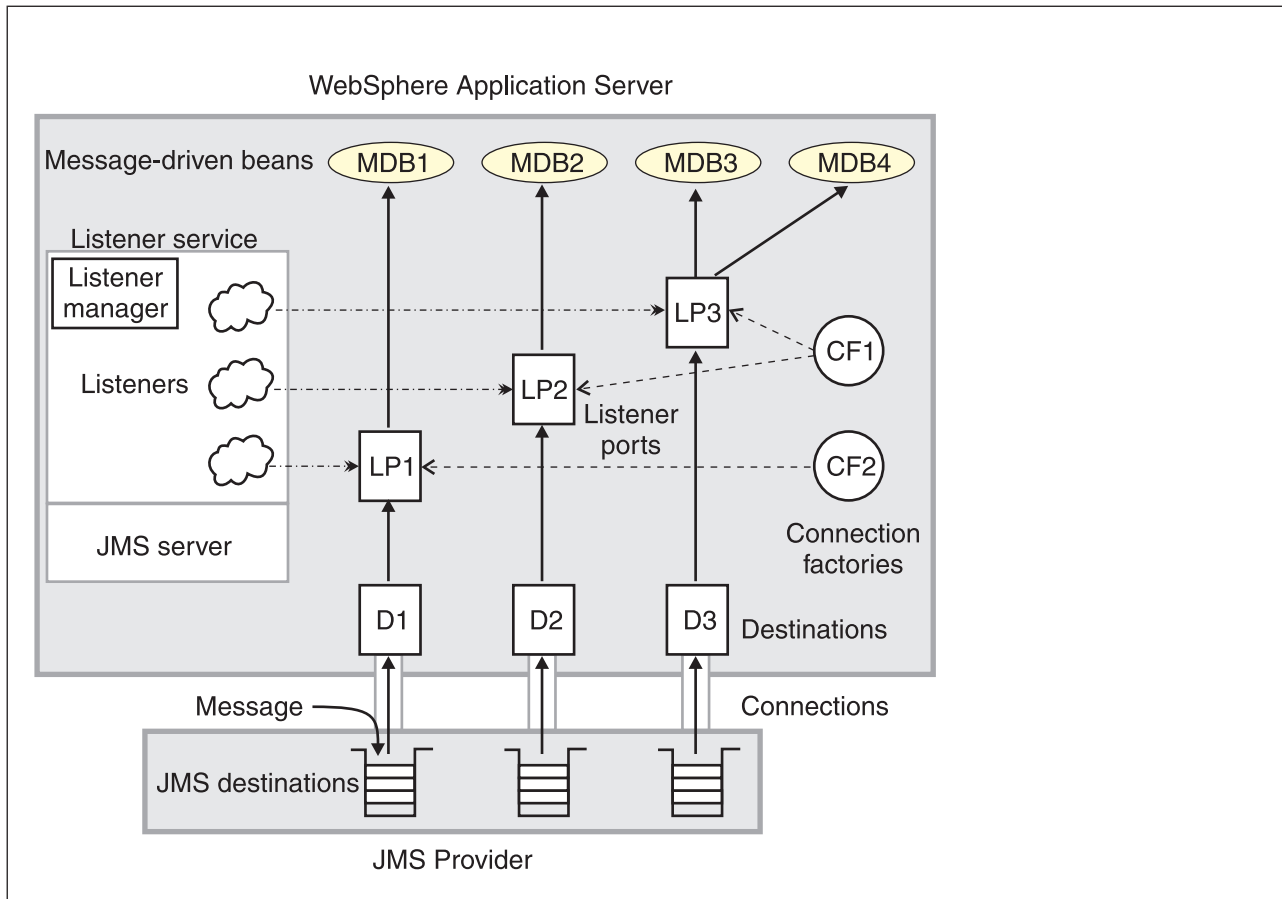


Figure 2. The main components for message-driven beans. This figure shows the main components of WebSphere support for message-driven beans, from JMS provider through a connection to a destination, listener port, then deployed message-driven bean that processes the message retrieved from the destination. Each listener port defines the association between a connection factory, destination, and a deployed message-driven bean. The other main components are the message listener service, which comprises a listener for each listener port, all controlled by the same listener manager. For more information, see the text that accompanies this figure.

The message listener service is an extension to the JMS functions of the JMS provider and provides a listener manager, which controls and monitors one or more JMS listeners.

Each listener monitors either a JMS queue destination (for point-to-point messaging) or a JMS topic destination (for publish/subscribe messaging).

A connection factory is used to create connections with the JMS provider for a specific JMS queue or topic destination. Each connection factory encapsulates the configuration parameters needed to create a connection to a JMS destination.

A listener port defines the association between a connection factory, a destination, and a deployed message-driven bean. Listener ports are used to simplify the administration of the associations between these resources.

When a deployed message-driven bean is installed, it is associated with a listener port and the listener for a destination. When a message arrives on the destination, the listener passes the message to a new instance of a message-driven bean for processing.

When an application server is started, it initializes the listener manager based on the configuration data. The listener manager creates a dynamic session thread pool

for use by listeners, creates and starts listeners, and during server termination controls the cleanup of listener message service resources. Each listener completes several steps for the JMS destination that it is to monitor, including:

- Creating a JMS server session pool, and allocating JMS server sessions and session threads for incoming messages.
- Interfacing with JMS ASF to create JMS connection consumers to listen for incoming messages.
- If specified, starting a transaction and requesting that it is committed (or rolled back) when the EJB method has completed.
- Processing incoming messages by invoking the `onMessage()` method of the specified enterprise bean.

Message-driven beans - transaction support

Message-driven beans can handle messages read from JMS destinations within the scope of a transaction. If transaction handling is specified for a JMS destination, the JMS listener starts a global transaction *before* it reads any incoming message from that destination. When the message-driven bean processing has finished, the JMS listener commits or rolls back the transaction (using JTA transaction control).

Note:

- All messages retrieved from a specific destination have the same transactional behavior.

If messages are queued to be sent within a global transaction they are sent when the transaction is committed. If the processing of a message causes the transaction to be rolled back, then the message that caused the bean instance to be invoked is left on the JMS destination.

You can configure the **Maximum retries** property of the listener port to define the maximum number of times the listener attempts to read a message from a destination. When the Max retries limit is reached, the listener for that destination is stopped. When you have resolved the problem, you must then restart the listener.

Designing an enterprise application to use message-driven beans

This topic describes things to consider when designing an enterprise application to use message-driven beans.

The considerations in this topic are based on a generic enterprise application that uses one message-driven bean to retrieve messages from a JMS queue destination and passes the messages on to another enterprise bean that implements the business logic.

To design an enterprise application to use message-driven beans, complete the following steps:

1. Identify the JMS resources that the application is to use. This helps to identify the properties of resources that need to be used within the application and configured as application deployment descriptors or within WebSphere Application Server.

JMS resource type	Properties
Queue connection factory	Name: SamplePtoPQueueConnectionFactory JNDI Name: Sample/JMS/QCF

JMS resource type	Properties
Queue destination	Name: Q1 JNDI Name: Sample/JMS/Q1
Listener port (for the destination)	Name: SamplePtoPListenerPort Connection Factory JNDI Name: Sample/JMS/QCF Destination JNDI Name: Sample/JMS/Q1 Maximum Sessions: 5 Maximum Retries: 10 Maximum Messages: 1
Message-driven bean (deployment properties)	Name: JMSppSampleMDBBean Transaction type: Container Destination type: Queue Listener port name: SamplePtoPListenerPort
Business logic bean	Name: MyLogicBean

Ensure that you use consistent values where needed; for example, the JNDI names for the connection factory and destination must be the same for both those resources and the equivalent properties of the listener port.

2. Separation of business logic. You are recommended to develop a message-driven bean to delegate the business processing of incoming messages to another enterprise bean. This provides clear separation of message handling and business processing. This also enables the business processing to be invoked by either the arrival of incoming messages or, for example, from a WebSphere J2EE client.
3. Security considerations.
Messages arriving at a destination being processed by a listener have no client credentials associated with them; the messages are anonymous. Security depends on the role specified by the RunAs Identity for the message-driven bean as an EJB component. For more information about EJB security, see EJB component security.
4. General JMS considerations For Publish/Subscribe messaging, choose the JMS server port to be used depending on your needs for transactions or performance:

Queued port

The TCP/IP port number of the listener port used for all point-to-point and Publish/Subscribe support.

Direct port

The TCP/IP port number of the listener port used for direct TCP/IP connection (non-transactional, non-persistent, and non-durable subscriptions only) for Publish/Subscribe support.

Note: Message-driven beans cannot use the direct listener port for Publish/Subscribe support. Therefore, any topic connection factory configured with **Portset** to Direct cannot be used with message-driven beans.

A non-durable subscriber can only be used in the same transactional context (for example, a global transaction or an unspecified transaction context) that existed when the subscriber was created. For more information about this context restriction, see The effect of transaction context on non-durable subscribers.

Developing an enterprise application to use message-driven beans

Use this task to develop an enterprise application to use a message-driven bean. The message-driven bean is invoked by a JMS listener when a message arrives on the input queue that the listener is monitoring.

You are recommended to develop the message-driven bean to delegate the business processing of incoming messages to another enterprise bean, to provide clear separation of message handling and business processing. This also enables the business processing to be invoked by either the arrival of incoming messages or, for example, from a WebSphere J2EE client. Responses can be handled by another enterprise bean acting as a sender bean, or handled in the message-driven bean.

You develop an enterprise application to use a message-driven bean like any other enterprise bean, except that a message-driven bean does not have a home interface or a remote interface.

This topic describes how to develop a completely new message-driven bean class. If you have a WAS 4.0 enterprise application that uses the JMS listener, you can migrate that application to use message-driven beans, as described in *Migrating a WAS 4.0 JMS listener application to use message-driven beans*.

This topic describes how to develop a completely new message-driven bean class. If you have a WAS 4.0 enterprise application that uses the JMS listener, you can migrate that application to use message-driven beans.

For more information about writing the message-driven bean class, see *Creating a message-driven bean* in the WebSphere Studio help bookshelf.

To develop an enterprise application to use a message-driven bean, complete the following steps:

1. Creating the Enterprise Application project, as described in the WebSphere Studio article *Creating an Enterprise Application project*.
2. Creating the message-driven bean class.

You can use the New Enterprise Bean wizard of WebSphere Studio Application Developer to create an enterprise bean with a bean type of Message-driven bean. The wizard creates appropriate methods for the type of bean.

By convention, the message bean class is named *nameBean*, where *name* is the name you assign to the message bean; for example:

```
public class MyJMSppMDBBean implements MessageDrivenBean, MessageListener
```

The message-driven bean class must define and implement the following methods:

- `onMessage(message)`, which must meet the following requirements:
 - The method must have a single argument of type `javax.jms.Message`.
 - The throws clause must *not* define any application exceptions.
 - If the message-driven bean is configured to use bean-managed transactions, it must call the `javax.transaction.UserTransaction` interface to scope the transactions. Because these calls occur inside the `onMessage()` method, the transaction scope does not include the initial message receipt. This means the application server is given one attempt to process the message.

To handle the message within the `onMessage()` method (for example, to pass the message on to another enterprise bean), you use standard JMS. (This is known as bean-managed messaging.)

- `ejbCreate()`

You must define and implement an `ejbCreate` method for each way in which you want a new instance of an enterprise bean to be created.

- `ejbRemove()`.

This method is invoked by the container when a client invokes the `remove` method inherited by the enterprise bean's home interface from the `javax.ejb.EJBHome` interface. This method must contain any code that you want to execute before an enterprise bean instance is removed from the container (and the associated data is removed from the data source).

For example, the following code extract shows how to access the text and the JMS MessageID, from a JMS message of type `TextMessage`:

The result of this step is a message-driven bean that can be assembled into an

```
public void onMessage(javax.jms.Message msg)
{
    String text      = null;
    String messageID = null;

    try
    {
        text = ((TextMessage)msg).getText();

        System.out.println("senderBean.onMessage(), msg text2: "+text);

        //
        // store the message id to use as the Correlator value
        //
        messageID = msg.getJMSMessageID();

        // Call a private method to put the message onto another queue
        putMessage(messageID, text);
    }
    catch (Exception err)
    {
        err.printStackTrace();
    }
    return;
}
```

Figure 3. Code example: The `onMessage()` method of a message bean. This figure shows a code extract for a basic `onMessage()` method of a sample message-driven bean. The method unpacks the incoming text message to extract the text and message identifier and calls a private `putMessage` method (defined within the same message bean class) to put the message onto another queue.

.EAR file for deployment.

3. Assembling and packaging the application for deployment.

You can use WebSphere Studio to assemble and package the application for deployment.

The result of this task is an .EAR file, containing an application message-driven bean, that can be deployed in WebSphere Application Server.

After you have developed an enterprise application to use message-driven beans, configure and deploy the application; for example, define the listener ports for the message-driven beans and, optionally, change the deployment descriptor attributes

for the application. For more information about configuring and deploying an application that uses message-driven beans, see [Deploying an enterprise application to use message-driven beans](#)

Deploying an enterprise application to use message-driven beans

Use this task to deploy an enterprise application to use message-driven beans.

This task description assumes that you have an .EAR file, which contains an application enterprise bean with code for message-driven beans, that can be deployed in WebSphere Application Server.

To deploy an enterprise application to use message-driven beans, complete the following steps:

1. Use the WebSphere administrative console to define the listener ports for the application, as described in [Adding a new listener port](#).
2. For each message-driven bean in the application, use the application assembly tool to configure the deployment attributes to match the listener port definitions, as described in [Configuring deployment attributes for a message-driven bean](#).
3. Use the WebSphere administrative console to install the application

This stage is a standard WebSphere Application Server task, as described in [Installing a new application](#).

When you install the application, you are prompted to specify the name of the listener port that the application is to use for late responses. Select the listener port, then click **OK**.

Configuring deployment attributes for a message-driven bean

Use this task to configure the message-driven beans deployment attributes for an enterprise bean, to override the deployment attributes defined within the application EAR file.

This task description assumes that you have an EAR file, which contains an application enterprise bean developed as a message-driven bean, that can be deployed in WebSphere Application Server.

Note: After deployment code has been generated for an application, the deployable archive is renamed with the prefix `Deployed_`. Any subsequent changes to the archive from within the Application Assembly Tool are applied to the version of the archive that existed prior to code generation. To see changes reflected in your application, you must regenerate deployment code and re-install the deployable archive.

To configure the message-driven beans deployment attributes for an enterprise bean, use the the application assembly tool to configure the deployment attributes of the application to match the listener port definitions:

1. Launch the Application Assembly Tool.
2. Create or edit the application EAR file. For example, to change attributes of an existing application, click **File-> Open** then select the the EAR file.
3. In the navigation pane, select the message-driven bean instance; for example, expand `ejb_module_instance-> Message-driven beans` then select the bean instance. A property dialog notebook for the message-driven bean is displayed in the property pane.

4. Specify general deployment properties.
 - a. In the property pane, select the General tab.
 - b. Specify the following properties:

Transaction type

Whether the message bean manages its own transactions or the container manages transactions on behalf of the bean. All messages retrieved from a specific destination have the same transactional behavior. To enable the transactional behavior that you want, you must configure the JMS destination with the same transactional behavior as you configure for the message bean.

Bean The message bean manages its own transactions

Container

The container manages transactions on behalf of the bean

5. Specify advanced deployment properties.
 - a. In the property pane, select the Advanced tab.
 - b. Specify the following properties:

Message selector

The JMS message selector to be used to determine which messages the message bean receives; for example:

```
JMSType='car' AND color='blue' AND weight>2500
```

The selector string can refer to fields in the JMS message header and fields in the message properties. Message selectors cannot reference message body values.

Acknowledge mode

How the session acknowledges any messages it receives.

This property applies only to message-driven beans that uses bean-managed transaction demarcation (**Transaction type** is set to Bean).

Auto Acknowledge

The session automatically acknowledges a message when it has either successfully returned from a call to receive, or the message listener it has called to process the message successfully returns.

Dups OK Acknowledge

The session lazily acknowledges the delivery of messages. This is likely to result in the delivery of some duplicate messages if JMS fails, so it should be used only by consumers that are tolerant of duplicate messages.

As defined in the EJB specification, clients cannot use using Message.acknowledge() to acknowledge messages. If a value of CLIENT_ACKNOWLEDGE is passed on the createxxxSession call, then messages are automatically acknowledged by the application server and Message.acknowledge() is not used.

Destination type

Whether the message bean uses a queue or topic destination.

Queue

The message bean uses a queue destination.

Topic The message bean uses a topic destination.

Subscription durability

Whether a JMS topic subscription is durable or non-durable.

Durable

A subscriber registers a durable subscription with a unique

identity that is retained by JMS. Subsequent subscriber objects with the same identity resume the subscription in the state it was left in by the earlier subscriber. If there is no active subscriber for a durable subscription, JMS retains the subscription's messages until they are received by the subscription or until they expire.

Nondurable

Non-durable subscriptions last for the lifetime of their subscriber object. This means that a client sees the messages published on a topic only while its subscriber is active. If the subscriber is not active, the client is missing messages published on its topic.

A non-durable subscriber can only be used in the same transactional context (for example, a global transaction or an unspecified transaction context) that existed when the subscriber was created. For more information about this context restriction, see *The effect of transaction context on non-durable subscribers*.

6. Specify bindings deployment properties.
 - a. In the property pane, select the Bindings tab.
 - b. Specify the following property:
 - Listener port name**
The name of the listener port for this message-driven bean.
7. To apply the changes and close the Application Assembly Tool, click **OK**. Otherwise, to apply the values but keep the property dialog open for additional edits, click **Apply**.
8. To see changes reflected in your application, regenerate deployment code and reinstall the deployable archive.

Configuring message listener resources for message-driven beans

Use the following tasks to configure resources needed by the message listener service to support message-driven beans.

- Configuring the message listener service
- Adding a new listener port
- Configuring a listener port
- Configuring security for message-driven beans

Configuring the message listener service

Use this task to configure the properties of the message listener service for an application server.

To configure the properties of the message listener service for an application server, use the administrative console to complete the following steps:

1. In the navigation pane, select **Servers-> Application Servers** This displays a table of the application servers in the administrative domain.
2. In the content pane, click the name of the application server. This displays the properties of the application server in the content pane.
3. In the Additional Properties table, select **Message Listener Service** This displays the Message Listener Service properties in the content pane.
4. Specify appropriate properties of the message listener service.
5. Click **OK**.

6. Save your configuration.
7. To have the changed configuration take effect, stop then restart the Application Server.

Message listener service

The message listener service is an extension to the JMS functions of the JMS provider. It provides a listener manager that controls and monitors one or more JMS listeners, which each monitor a JMS destination on behalf of a deployed message-driven bean.

This panel displays links to the Additional Properties pages for Listener Ports, Thread Pool, and Custom Properties for the message listener service.

To view this administrative console page, click **Servers-> *application_server*-> Message Listener Service**

Name:

The name by which the message listener service is known for administrative purposes.

Data type	String
Default	MsgLService

Description:

A description of the message listener service, for administrative purposes

Data type	String
Default	Null

Thread pool:

Controls the maximum number of threads the Message Listener Service is allowed to run. Select this link to display the service thread pool properties.

Adjust this parameter when multiple message-driven beans are deployed in the same application server and the sum of their maximum session values exceeds the default value of 10.

Data type	Integer
Units	Not applicable
Default	Minimum: 10, maximum 50
Range	Not applicable
Recommended	Set the minimum to the sum of all message-driven beans maximum session values. Set the maximum to anything equal or greater than the minimum.

Custom Properties:

Custom properties of the message listener service.

You can use the Custom properties page to define special properties for use by the message listener service.

- NON.ASF.RECEIVE.TIMEOUT

- MQJMS.POOLING.TIMEOUT
- MQJMS.POOLING.THRESHOLD
- MAX.RECOVERY.RETRIES
- RECOVERY.RETRY.INTERVAL

NON.ASF.RECEIVE.TIMEOUT:

The JMS server has two modes of operation, Application Server Facilities (ASF) and non-Application Server Facilities. ASF is meant to provide concurrency and transactional support for applications. Non-ASF mode is still transactional, but the path length is streamlined to improve performance.

Use Non-ASF if:

- Message order is a strict requirement
- Concurrent PTP messages are desired
- The property `NON.ASF.RECEIVE.TIMEOUT` exists and has a value greater than 0

Do not use Non-ASF if concurrent publications and subscriptions messages are desired because ASF provides better throughput.

Data type	Integer
Units	Milliseconds
Default	ASF mode (custom property not created)
Range	
Recommended	The value is the number of milliseconds it takes for a message to be delivered. If a timeout occurs, it must recycle causing extra work. This should be set to lower than the transaction timeout, but close to it with several (10 or more) seconds to spare. The seconds to spare should be more if under extreme loads in which threads are waiting long periods of time to get CPU cycles.

MQJMS.POOLING.TIMEOUT:

The number of milliseconds after which a connection in the pool is destroyed if it has not been used.

An `MQSimpleConnectionManager` allocates connections on a most-recently-used basis, and destroys connections on a least-recently-used basis. By default, a connection is destroyed if it has not been used for five minutes.

Data type	Integer
Units	Milliseconds
Default	5 minutes
Range	

MQJMS.POOLING.THRESHOLD:

The maximum number of unused connections in the pool.

An `MQSimpleConnectionManager` allocates connections on a most-recently-used basis, and destroys connections on a least-recently-used basis. By default, a connection is destroyed if there are more than ten unused connections in the pool.

Data type	Integer
Units	Number of connections
Default	10
Range	

MAX.RECOVERY.RETRIES:

The maximum number of times that the listener service tries to get a message from a listener port before the associated listener is stopped, in the range 0 through 2147483647.

Data type	Integer
Units	Retry attempts
Default	0 (no retries)
Range	0 (no retries) through 2147483647

RECOVERY.RETRY.INTERVAL:

The time in seconds between retry attempts by the listener service to get a message from a listener port.

Data type	Integer
Units	Seconds
Default	10
Range	1 through 2147483647

Message listener service custom properties:

Use this panel to view or change an optional set of name and value pairs for custom properties of the message listener service.

To view this administrative console page, click **Servers-> application_server-> Message Listener Service-> (In content pane, under Additional Properties) Custom Properties**

You can use the Custom properties page to define the following properties for use by the message listener service.

- NON.ASF.RECEIVE.TIMEOUT
- MQJMS.POOLING.TIMEOUT
- MQJMS.POOLING.THRESHOLD
- MAX.RECOVERY.RETRIES
- RECOVERY.RETRY.INTERVAL

NON.ASF.RECEIVE.TIMEOUT:

The JMS server has two modes of operation, Application Server Facilities (ASF) and non-Application Server Facilities. ASF is meant to provide concurrency and transactional support for applications. Non-ASF mode is still transactional, but the path length is streamlined to improve performance.

Use Non-ASF if:

- Message order is a strict requirement
- Concurrent PTP messages are desired

- The property `NON.ASF.RECEIVE.TIMEOUT` exists and has a value greater than 0

Do not use Non-ASF if concurrent publications and subscriptions messages are desired because ASF provides better throughput.

Data type	Integer
Units	Milliseconds
Default	ASF mode (custom property not created)
Range	
Recommended	The value is the number of milliseconds it takes for a message to be delivered. If a timeout occurs, it must recycle causing extra work. This should be set to lower than the transaction timeout, but close to it with several (10 or more) seconds to spare. The seconds to spare should be more if under extreme loads in which threads are waiting long periods of time to get CPU cycles.

MQJMS.POOLING.TIMEOUT:

The number of milliseconds after which a connection in the pool is destroyed if it has not been used.

An `MQSimpleConnectionManager` allocates connections on a most-recently-used basis, and destroys connections on a least-recently-used basis. By default, a connection is destroyed if it has not been used for five minutes.

Data type	Integer
Units	Milliseconds
Default	5 minutes
Range	

MQJMS.POOLING.THRESHOLD:

The maximum number of unused connections in the pool.

An `MQSimpleConnectionManager` allocates connections on a most-recently-used basis, and destroys connections on a least-recently-used basis. By default, a connection is destroyed if there are more than ten unused connections in the pool.

Data type	Integer
Units	Number of connections
Default	10
Range	

MAX.RECOVERY.RETRIES:

The maximum number of times that the listener service tries to get a message from a listener port before the associated listener is stopped, in the range 0 through 2147483647.

Data type	Integer
Units	Retry attempts
Default	0 (no retries)

Range 0 (no retries) through 2147483647

RECOVERY.RETRY.INTERVAL:

The time in seconds between retry attempts by the listener service to get a message from a listener port.

Data type	Integer
Units	Seconds
Default	10
Range	1 through 2147483647

Message listener port collection:

The message listener ports configured in the administrative domain

This panel displays a list of the message listener ports configured in the administrative domain. Each listener port is used with a message-driven bean to automatically receive messages from an associated JMS destination. You can use this panel to add new listener ports or to change the properties of existing listener ports. For more information about the property fields for listener ports, see Listener port properties.

To view this administrative console page, click **Servers-> application_server-> Message Listener Service-> Listener Ports**

Listener port settings:

A listener port is used to simplify administration of the association between a connection factory, destination, and deployed message-driven bean.

Use this panel to view or change the configuration properties of the selected listener port.

To view this administrative console page, click **Servers-> application_server-> Message Listener Service-> Listener Ports-> listener_port**

Name:

The name by which the listener port is known for administrative purposes.

Data type	String
Default	Null

Initial state:

The state that you want the listener port to have when the application server is next restarted

Data type	Enum
Units	Not applicable
Default	Started

Range

Started When the application server is next started, the listener port is started automatically.

Stopped

When the application server is next started, the listener port is not started automatically. If message-driven beans are to use this listener port on the application server, the system administrator must start the port manually or select the Started value of this property then restart the application server.

Description:

A description of the listener port, for administrative purposes within IBM WebSphere Application Server.

Data type	String
Default	Null

Connection factory JNDI name:

The JNDI name for the JMS connection factory to be used by the listener port; for example, `jms/connFactory1`.

Data type	String
Default	Null

Destination JNDI name:

The JNDI name for the destination to be used by the listener port; for example, `jms/destn1`.

If the extended messaging service is to use this listener port to handle late responses, the value of this property must match the JMS response destination on the output port used by the sender bean. Also cannot use a temporary destination for late responses.

Data type	String
Default	Null

Maximum sessions:

Specifies the maximum number of concurrent JMS server sessions used by a listener to process messages.

Each session corresponds to a single listener thread, but does not control the number of concurrently processed messages. Adjust this parameter when the machine running the process application does not realize the available capacity and produces less throughput by running long processes.

This parameter helps to add some level of concurrency so that messages are received concurrently before the synchronized part of receiving the message occurs. A 40% increase in throughput is observed in a process application with long running processes on a NetFinity 5500 500 MHz, 4-way, 4GB RAM system.

An alternative to making this change in the administrative console is to change the maximum sessions value in the `server.xml` file under the listener ports stanza as follows:

```
<listenerPorts xmi:id="ListenerPort_1" name="bpeIntListenerPort"
description="Internal Listener Port for Process Choreographer"
connectionFactoryJNDIName="jms/bpeCF" destinationJNDIName="jms/bpeIntQueue"
maxSessions="5" maxRetries="10" maxMessages="1">
<stateManagement xmi:id="StateManageable_5" initialState="START" />
</listenerPorts>
```

Data type	Integer
Units	Sessions
Default	1
Range	1 through 2147483647
Recommended	If message concurrency, that is multiple messages processed simultaneously, is desired, set the value to 2-4 sessions per system processor. Keep the value as low as possible to eliminate client thrashing. If a strict message order is desired, set the value to 4 so there is always a thread waiting in a hot state, blocked on receiving the message.

Maximum retries:

The maximum number of times that the listener tries to deliver a message before the listener is stopped, in the range 0 through 2147483647.

The maximum number of times that the listener tries to deliver a message to a message-driven bean instance before the listener is stopped.

Data type	Integer
Units	Retry attempts
Default	0 (no retries)
Range	0 (no retries) through 2147483647

Maximum messages:

The maximum number of messages that the listener can process in one JMS server session.

Adjust this parameter when message order is not a strict requirement. This parameter controls the concurrency level in the listener service.

Data type	Integer
Units	Number of messages
Default	1
Range	1 through 2147483647

Recommended

If message concurrency, that is multiple messages processed simultaneously, is desired, set the value to 2-4 sessions per system processor. Keep the value as low as possible to eliminate client thrashing. If a strict message order is desired, set the value to 1.

Adding a new listener port

Use this task to add a new listener port to the message listener service, so that message-driven beans can be associated with the port to retrieve messages.

To add a new listener port, use the administrative console to complete the following steps:

1. In the navigation pane, select **Servers-> Application Servers** This displays a table of the application servers in the administrative domain.
2. In the content pane, click the name of the application server. This displays the properties of the application server in the content pane.
3. In the Additional Properties table, select **Message Listener Service** This displays the Message Listener Service properties in the content pane.
4. In the content pane, select **Listener Ports**. This displays a list of the listener ports.
5. In the content pane, click **New**.
6. Specify appropriate properties for the listener port.
7. Click **OK**.
8. To save your configuration, click **Save** on the task bar of the Administrative console window.
9. To have the changed configuration take effect, stop then restart the application server.

If enabled, the listener port is started automatically when a message-driven bean associated with that port is installed.

Configuring a listener port

Use this task to change the properties of an existing listener port, used by message-driven beans associated with the port to retrieve messages.

To configure the properties of a listener port, use the administrative console to complete the following steps:

1. In the navigation pane, select **Servers-> Application Servers** This displays a table of the application servers in the administrative domain.
2. In the content pane, click the name of the application server. This displays the properties of the application server in the content pane.
3. In the Additional Properties table, select **Message Listener Service** This displays the Message Listener Service properties in the content pane.
4. In the content pane, click **Listener Ports**. This displays a list of the listener ports.
5. Click the listener port that you want to modify. This displays the properties of the listener port in the content pane.
6. Specify appropriate properties for the listener port.

7. Click **OK**.
8. To save your configuration, click **Save** on the task bar of the Administrative console window.
9. To have the changed configuration take effect, stop then restart the application server.

Deleting a listener port

Use this task to delete a listener port from the message listener service, to prevent message-driven beans associated with the port from retrieving messages.

To delete a listener port, use the administrative console to complete the following steps:

1. In the navigation pane, select **Servers-> Application Servers** This displays a table of the application servers in the administrative domain.
2. In the content pane, click the name of the application server. This displays the properties of the application server in the content pane.
3. In the Additional Properties table, select **Message Listener Service** This displays the Message Listener Service properties in the content pane.
4. In the content pane, select **Listener Ports**. This displays a list of the listener ports.
5. In the content pane, select the checkbox for the listener port that you want to delete.
6. Click **Delete**. This action stops the port (needed to allow the port to be deleted) then deletes the port.
7. To save your configuration, click **Save** on the task bar of the Administrative console window.
8. To have the changed configuration take effect, stop then restart the application server.

Configuring security for message-driven beans

Use this task to configure resource security and security permissions for message-driven beans.

Messages arriving at a listener port have no client credentials associated with them. The messages are anonymous.

To call secure enterprise beans from a message-driven bean, the message-driven bean needs to be configured with a RunAs Identity deployment descriptor. Security depends on the role specified by the RunAs Identity for the message-driven bean as an EJB component.

For more information about EJB security, see EJB component security. For more information about configuring security for your application, see Assembling secured applications.

JMS connections used by message-driven beans can benefit from the added security of using J2C container-managed authentication. To enable the use of J2C container authentication aliases and mapping, define a J2C container-managed alias on the JMS connection factory definition that the MDB is using to listen upon (defined by the **Connection factory JNDI name** property of the listener port). If defined, the listener uses the container-managed authentication alias for its

JMSCConnection security credentials instead of any application-managed alias. To set the container-managed alias, use the administrative console to complete the following steps:

1. To display the listener port settings, click **Servers-> application_server-> Message Listener Service-> Listener Ports-> listener_port**
2. To get the name of the JMS connection factory, look at the **Connection factory JNDI name** property.
3. Display the JMS connection factory properties. For example, to display the properties of a queue connection factory provided by the embedded WebSphere JMS provider, click **Resources-> WebSphere JMS Provider-> (In content pane, under Additional Properties) WebSphere Queue Connection Factories-> connection_factory**
4. Set the **Container-managed Authentication Alias** property.

Administering listener ports

Use the following tasks to administer listener ports, which each define the association between a connection factory, a destination, and a message-driven bean.

You can use the WebSphere administrative console to administer listener ports, as described in the following tasks.

- Adding a new listener port
Use this task to create a new listener port, to specify a new association between a connection factory, a destination, and a message-driven bean. This enables deployed message-driven beans associated with the port to retrieve messages from the destination.
- Configuring a listener port
Use this task to view or change the configuration properties of a listener port.
- Starting a listener port
Use this task to start a listener port manually.
- Stopping a listener port
Use this task to stop a listener port manually.

Note: If configured as enabled, a listener port is started automatically when a message-driven bean associated with that port is installed. You do not normally need to start or stop a listener port manually.

Starting a listener port

Use this task to start a listener port on an application server, to enable the listeners for message-driven beans associated with the port to retrieve messages.

A listener is active, that is able to receive messages from a destination, if the deployed message-driven bean, listener port, and message listener service are all started. Although you can start these components in any order, they must all be in a started state before the listener can retrieve messages.

If configured as enabled, a listener port is started automatically when a message-driven bean associated with that port is installed. However, you can start a listener port manually, as described in this topic.

When a listener port is started, the listener manager tries to start the listeners for each message-driven bean associated with the port. If a message-driven bean is stopped, the port is started but the listener is not started, and remains stopped. If you start a message-driven bean, the related listener is started.

To start a listener port on an application server, use the administrative console to complete the following steps:

1. If you want the listener for a deployed message-driven bean to be able to receive messages at the port, check that the message-driven bean has been started.
2. In the navigation pane, select **Servers-> Application Servers** This displays a table of the application servers in the administrative domain.
3. In the content pane, click the name of the application server. This displays the properties of the application server in the content pane.
4. In the Additional Properties table, select **Message Listener Service** This displays the Message Listener Service properties in the content pane.
5. In the content pane, select **Listener Ports**. This displays a list of the listener ports.
6. Select the checkbox for the listener port that you want to start.
7. Click **Start**.
8. To save your configuration, click **Save** on the task bar of the Administrative console window.

Stopping a listener port

Use this task to stop a listener port on an application server, to prevent the listeners for message-driven beans associated with the port from retrieving messages.

When you stop a listener port as described in this topic, the listener manager stops the listeners for all message-driven beans associated with the port.

To stop a listener port on an application server, use the administrative console to complete the following steps:

1. In the navigation pane, select **Servers-> Application Servers** This displays a table of the application servers in the administrative domain.
2. In the content pane, click the name of the application server. This displays the properties of the application server in the content pane.
3. In the Additional Properties table, select **Message Listener Service** This displays the Message Listener Service properties in the content pane.
4. In the content pane, select **Listener Ports**. This displays a list of the listener ports.
5. In the content pane, select the listener port that you want to stop.
6. Click **Stop**.
7. To save your configuration, click **Save** on the task bar of the Administrative console window.
8. To have the changed configuration take effect, stop then restart the application server.

Important files for message-driven beans and extended messaging

The following files in the WAS_HOME/temp directory are important for the operation of the WebSphere Application Server messaging service, so should not be deleted. If you do need to delete the WAS_HOME/temp directory or other files in it, ensure that you preserve the following files.

server_name-durableSubscriptions.ser

You should not delete this file, because the messaging service uses it to keep track of durable subscriptions for message-driven beans. If you

uninstall an application that contains a message-driven bean, this file is used to unsubscribe the durable subscription.

server_name-AsyncMessageRequestLog.ser

You should not delete this file, because the messaging service uses it to keep track of late responses that need to be delivered to the late response message handler for the extended messaging provider.

Troubleshooting message-driven beans

Use this overview task to help resolve a problem that you think is related to message-driven beans.

Message-driven beans support uses the standard WebSphere Application Server troubleshooting facilities. If you encounter a problem that you think might be related to the message-driven beans, complete the following stages:

1. Check for messages about message-driven beans in the application server's SystemOut log at *was_home\logs\server\SystemOut*. Look in the SystemOut log for messages that indicate a problem with JMS resources for message-driven beans, such as listener ports.
2. Check for more messages in the application server's SystemOut log. If the JMS server is running, but you have problems accessing JMS resources, check the SystemOut log file, which should contain more error messages and extra details about the problem.
3. Check the Release Notes for specific problems and workarounds The section *Possible Problems and Suggested Fixes* of the Release Notes, available from the WebSphere Application Server library web site, is updated regularly to contain information about known defects and their workarounds. Check the latest version of the Release Notes for any information about your problem. If the Release Notes does not contain any information about your problem, you can also search the Technotes database on the WebSphere Application Server web site.
4. Check that message listener service has started. The message listener service is an extension to the JMS functions of the JMS provider. It provides a listener manager that controls and monitors one or more JMS listeners, which each monitor a JMS destination on behalf of a deployed message-driven bean.
5. Check your JMS resource configurations If the WebSphere Messaging functions seem to be running properly (the JMS server is running without problems), check that the JMS resources have been configured correctly. For example, check that the listener ports have been configured correctly and have been started.
6. Check for problems with the WebSphere Messaging functions For more information about troubleshooting WebSphere Messaging, see the related topics.
7. Get a detailed exception dump for messaging. If the information obtained in the preceding steps is still inconclusive, you can enable the application server debug trace for the "Messaging" group to provide a detailed exception dump.

Message-driven beans samples

The following examples are provided, as part of the WebSphere Samples Gallery, to illustrate use of the message-driven beans support. When the Samples are installed on your local machine, they are available to try out. Locate them at <http://localhost:9080/WSsamples/> <http://localhost:9080/WSsamples/>. (The default port is 9080.) For more information about where to find the Samples Gallery, see Samples Gallery.

- Point-to-point samples:
 - "Tutorial: Creating JMS message sample"

This tutorial is designed to help you develop and deploy a JMS message sample application that tests the WebSphere Application Server message-driven beans support in a point-to-point scenario. This sample illustrates how to develop and deploy an application that comprises the following components:

 - A Java/JMS program that writes a message to a queue.
 - A message-driven bean that is invoked by a JMS listener when a message arrives on a defined queue.

For more information about this sample, see the samples article "Tutorial: Creating JMS message sample" that is installed with the Samples option.
 - "Sample: Message Listener (point-to-point)"

This sample is designed to demonstrate the use and behavior of message-driven beans for a simple point-to-point scenario. This sample uses the JMS message sample deployed in the sample above.

For more information about this sample, see the samples article "Sample: Message Listener (Point-to-Point)" that is installed with the Samples option.
- Publish/subscribe samples
 - "Tutorial: Creating JMS message publish/subscribe sample"

This tutorial is designed to help you develop and deploy a JMS message sample application that tests the WebSphere Application Server message-driven beans support in a publish/subscribe scenario. This sample illustrates how to develop and deploy an application that comprises the following components:

 - A client program that starts the message sequence by publishing a message to a selected topic.
 - A message-driven bean that is invoked by a JMS listener when the broker passes a message to the listener from a topic to which it has subscribed.

For more information about this sample, see the samples article "Tutorial: Creating JMS message publish/subscribe sample" that is installed with the Samples option.
 - "Sample: Message Listener (publish/subscribe)"

This sample is designed to demonstrate the use and behavior of message-driven beans for a simple publish/subscribe scenario. This sample uses the JMS message sample deployed in the publish/subscribe sample above.

For more information about this sample, see the samples article "Sample: Message Listener (publish/subscribe)" that is installed with the Samples option.

Chapter 6. Using application clients

An application client module is a JAR (Java ARchive) file containing a client for accessing a Java application.

1. Decide on a type of application client.
2. Develop the application client code.
 - Develop ActiveX application client code.
 - Develop applet client code.
 - Develop J2EE application client code.
 - Develop pluggable application client code.
 - Develop thin application client code.
3. Assemble the application client.
4. Deploy the application client.
 - Deploy the application client on z/OS.
 - Deploy the application client on Windows.
5. Run the application client on z/OS or OS/390.

View the Samples gallery for more information about application clients. Before you run the basicCalculator Sample, ensure the JMS Server is started.

Application clients

In a traditional client server environment, the client requests a service and the server fulfills the request. Multiple clients use a single server. Clients can also access several different servers. This model persists for Java clients except now these requests make use of a client run-time environment.

In this model, the client application requires a servlet to communicate with the enterprise bean, and the servlet must reside on the same machine as the WebSphere Application Server.

With WebSphere Application Server Version 5, application clients now consist of the following models:

- ActiveX application client
- Applet client
- J2EE application client
- Pluggable application client
- Thin application client

WebSphere Application Server for z/OS and OS/390 supports only two models:

- J2EE application client
- Pluggable application client

The *ActiveX application client* model, uses the Java Native Interface (JNI) architecture to programmatically access the Java virtual machine (JVM) API. Therefore the JVM code exists in the same process space as the ActiveX application (Visual Basic, VBScript, or Active Server Pages (ASP)) and remains attached to the process until that process terminates.

In the *Applet client* model, a Java applet embeds in a HyperText Markup Language (HTML) document residing on a remote client machine from the WebSphere Application Server. With this type of client, the user accesses an enterprise bean in the WebSphere Application Server through the Java applet in the HTML document.

The *J2EE application client* is a Java application program that accesses enterprise beans, Java Database Connectivity (JDBC), and Java Message Service message queues. The J2EE application client program runs on client machines. This program follows the same Java programming model as other Java programs; however, the J2EE application client depends on the application client run time to configure its execution environment, and uses the Java Naming and Directory Interface (JNDI) name space to access resources.

The *Pluggable and thin application clients* provide a lightweight Java client programming model. These clients are best suited in situations where a Java client application exists but the application needs enhancements to use enterprise beans, or where the client application requires a thinner, more lightweight environment than the one offered by the J2EE application client. The difference between the thin application client and the pluggable application client is that the thin application client includes a Java virtual machine (JVM) API, and the pluggable application client requires the user to provide this code. The pluggable application client uses the Sun Java Development Kit, and the thin application client uses the IBM Developer Kit For the Java Platform.

The J2EE application client programming model provides the benefits of the J2EE platform for the Java client application. The J2EE application client offers the ability to seamlessly develop, assemble, deploy and launch a client application. The tooling provided with the WebSphere platform supports the seamless integration of these stages to help the developer create a client application from start to finish.

When you develop a client application using and adhering to the J2EE platform, you can put the client application code from one J2EE platform implementation to another. The client application package can require redeployment using each J2EE platform deployment tool, but the code that comprises the client application does not change.

The application client run time supplies a container that provides access to system services for the client application code. The client application code must contain a main method. The application client run time invokes this main method after the environment initializes and runs until the Java virtual machine code terminates.

The J2EE platform allows the application client to use *nicknames* or *short names*, defined within the client application deployment descriptor. These deployment descriptors identify enterprise beans or local resources (JDBC, Java Message Service (JMS), JavaMail and URL APIs) for simplified resolution through JNDI use. This simplified resolution to the enterprise bean reference and local resource reference also eliminates changes to the client application code, when the underlying object or resource either changes or moves to a different server. When these changes occur, the application client can require redeployment. Although you can edit deployment descriptor files, do not use the administrative console to modify them.

The application client also provides initialization of the run-time environment for the client application. The deployment descriptor defines this unique initialization for each client application. The application client run time also provides support for security authentication to the enterprise beans and local resources.

The application client uses the RMI-IIOP protocol. Using this protocol enables the client application to access enterprise bean references and to use CORBA services provided by the J2EE platform implementation. Use of the RMI-IIOP protocol and the accessibility of CORBA services assist users in developing a client application that requires access to both enterprise bean references and CORBA object references.

When you combine the J2EE and CORBA environments or programming models in one client application, you must understand the differences between the two programming models to use and manage each appropriately.

View the Samples gallery for more information about application clients. Before you run the basicCalculator Sample, ensure the JMS Server is started.

Application client functions

Use the following table to identify the available functions in the different types of clients.

Note: WebSphere Application Server for z/OS supports only two types of application client:

- J2EE client
- Pluggable client

Available functions	ActiveX client	Applet client	J2EE client	Pluggable client	Thin client
Provides all the benefits of a J2EE platform	Yes	No	Yes	No	No
Portable across all J2EE platforms	No	No	Yes	No	No
Provides the necessary run-time to support communication between client and server	Yes	Yes	Yes	Yes	Yes
Allows the use of nicknames in the deployment descriptor files. Note: Although you can edit deployment descriptor files, do not use the administrative console to modify them.	Yes	No	Yes	No	No
Supports use of the RMI-IIOP protocol	Yes	Yes	Yes	Yes	Yes
Browser based application	No	Yes	No	No	No

Enables development of client applications that can access enterprise bean references and CORBA object references	Yes	Yes	Yes	Yes	Yes
Enables the initialization of the client application run-time environment	Yes	No	Yes	No	No
Supports security authentication to enterprise beans	Yes	Limited	Yes	Yes	Yes
Supports security authentication to local resources	Yes	No	Yes	No	No
Requires distribution of application to client machines	Yes	No	Yes	Yes	Yes
Enables access to enterprise beans and other Java classes through Visual Basic, VBScript, and Active Server Pages (ASP) code	Yes	No	No	No	No
Provides a lightweight client suitable for download	No	Yes	No	Yes	Yes
Enables access to Java Naming and Directory Interface (JNDI) for enterprise bean resolution	Yes	Yes	Yes	Yes	Yes
Runs on client machines that use the Sun Java Runtime Environment	No	No	No	Yes	No
Supports CORBA services (using CORBA services can render the application client code nonportable)	No	No	Yes	No	No

J2EE application clients

The J2EE application client programming model provides the benefits of Java™ 2 Platform for WebSphere Application Server Enterprise(J2EE).

The J2EE platform offers the ability to seamlessly develop, assemble, deploy and launch a client application. The tooling provided with the WebSphere platform supports the seamless integration of these stages to help the developer create a client application from start to finish.

When you develop a client application using and adhering to the J2EE platform, you can put the client application code from one J2EE platform implementation to another. The client application package can require redeployment using each J2EE platform deployment tool, but the code that comprises the client application does not change.

The J2EE application client run time supplies a container that provides access to system services for the application client code. The J2EE application client code must contain a main method. The J2EE application client run time invokes this main method after the environment initializes and runs until the Java virtual machine application terminates.

Application clients can use *nicknames* or *short names*, defined within the client application deployment descriptor with the J2EE platform. These deployment descriptors identify enterprise beans or local resources (Java Database Connectivity (JDBC), Java Message Service (JMS), JavaMail and URL APIs) for simplified resolution through JNDI use. This simplified resolution to the enterprise bean reference and local resource reference also eliminates changes to the application client code, when the underlying object or resource either changes or moves to a different server. When these changes occur, the application client can require redeployment. Although you can edit deployment descriptor files, do not use the administrative console to modify them.

The J2EE application client also provides initialization of the run-time environment for the client application. The deployment descriptor defines this unique initialization for each client application. The J2EE application client run time also provides support for security authentication to the enterprise beans and local resources.

The J2EE application client uses the RMI-IIOP protocol. Using this protocol enables the client application to access enterprise bean references and to use CORBA services provided by the J2EE platform implementation. Use of the RMI-IIOP protocol and the accessibility of CORBA services assist users in developing a client application that requires access to both enterprise bean references and CORBA object references.

When you combine the J2EE and the CORBA WebSphere Application Server Enterprise environments or programming models in one client application, you must understand the differences between the two programming models to use and manage each appropriately.

Pluggable application clients

The pluggable application client provides a lightweight, downloadable Java application run time capable of interacting with enterprise beans.

The pluggable application client requires that you have previously installed the Sun Java Runtime Environment (JRE) files. In all other aspects, the pluggable application client, and the thin application client are similar.

Note: *The pluggable client is only available on the Windows platform.*

This client is designed to support those users who want a lightweight Java client application programming environment, without the overhead of the J2EE platform on the client machine. The programming model for this client is heavily influenced by the CORBA programming model, but supports access to enterprise beans.

When accessing enterprise beans from this client, the client application can consider the enterprise beans object references as CORBA object references.

Tooling does not exist on the client; however, tooling does exist on the server. You are responsible for developing the client application, generating the necessary client bindings for the enterprise bean and CORBA objects, and after bundling these pieces together, installing them on the client machine.

The pluggable application client provides the necessary run time to support the communication needs between the client and the server.

The pluggable application client uses the RMI-IIOP protocol. Using this protocol enables the client application to access not only enterprise bean references and CORBA object references, but the protocol also allows the client application to use any supported CORBA services. Using the RMI-IIOP protocol along with the accessibility of CORBA services can assist a user in developing a client application that needs to access both enterprise bean references and CORBA object references.

When you combine the J2EE and CORBA environments in one client application, you must understand the differences between the two programming models to use and manage each appropriately.

The pluggable application client run time provides the necessary support for the client application for object resolution, security, Reliability Availability and Serviceability (RAS), and other services. However, this client does not support a container that provides easy access to these services. For example, no support exists for using *nicknames* for enterprise beans or local resource resolution. When resolving to an enterprise bean (using either Java Naming and Directory Interface (JNDI) or CosNaming) sources, the client application must know the location of the name server and the fully qualified name used when the reference was bound into the name space. When resolving to a local resource, the client application cannot resolve to the resource through a JNDI lookup. Instead the client application must explicitly create the connection to the resource using the appropriate API (JDBC, Java Message Service (JMS), and so on). This client does not perform initialization of any of the services that the client application might require. For example, the client application is responsible for the initialization of the naming service, either through CosNaming or JNDI APIs.

The pluggable application client offers access to most of the available client services in the J2EE application client. However, you cannot access the services in the pluggable client as easily as you can in the J2EE application client. The J2EE client has the advantage of performing a simple Java Naming and Directory Interface (JNDI) name space lookup to access the desired service or resource. The pluggable client must code explicitly for each resource in the client application. For example, looking up an enterprise bean Home requires the following code in a J2EE application client:


```

java.lang.Object ejbHome =
    initialContext.lookup("java:/comp/env/ejb/MyEJBHome");
MyEJBHome =
    (MyEJBHome)javax.rmi.PortableRemoteObject.narrow(ejbHome, MyEJBHome.class);

```

However, you need more explicit code in a Java pluggable application client:

```

java.lang.Object ejbHome =
    initialContext.lookup(
        "the/fully/qualified/path/to/actual/home/in/namespace/MyEJBHome");
MyEJBHome =
    (MyEJBHome)javax.rmi.PortableRemoteObject.narrow(ejbHome, MyEJBHome.class);

```

In this example, the J2EE application client accesses a logical name from the `java:/comp` name space. The J2EE client run time resolves that name to the physical location and returns the reference to the client application. The pluggable client must know the fully qualified physical location of the enterprise bean Home in the name space. If this location changes, the pluggable client application must also change the value placed on the `lookup()` statement.

In the J2EE client, the client application is protected from these changes because it uses the logical name. A change can require a redeployment of the EAR file, but the actual client application code remains the same.

The pluggable application client is a traditional Java application that contains a *main* function. The WebSphere pluggable application client provides run time support for accessing remote enterprise beans, and provides the implementation for various services (security, Workload Management (WLM), and others). This client can also access CORBA objects and CORBA based services. When using both environments in one client application, you need to understand the differences between the enterprise bean and CORBA programming models to manage both environments.

For instance, the CORBA programming model requires the CORBA CosNaming name service for object resolution in a name space. The enterprise beans programming model requires the JNDI name service. The client application must initialize and properly manage these two naming services.

Another difference applies to the enterprise bean model. Use the Java Naming and Directory Interface (JNDI) implementation in the enterprise bean model to initialize the Object Request Broker (ORB). The client application is unaware that an ORB is present. The CORBA model, however, requires the client application to explicitly initialize the ORB through the `ORB.init()` static method.

The pluggable application client provides a batch command that you can use to set the `CLASSPATH` and `JAVA_HOME` environment variables to enable the pluggable application client run time.

Developing J2EE application client code

A *J2EE application client* program operates similarly to a standard J2EE program in that it runs its own Java Virtual Machine code and is invoked at its main method.

A *J2EE application client* program operates similarly to a standard J2EE program in that it runs its own ASCII Java Virtual Machine code and is invoked at its main method. This JVM run-time environment is part of the client container, which provides the following services for the application client:

- Security
- Communications protocol support (for RMI/IIOP, HTTP, and so on)
- Naming support

The Java Virtual Machine application client program differs from a standard Java program because it uses the Java Naming and Directory Interface (JNDI) name space to access resources. In a standard Java program, the resource information is coded in the program.

Storing the resource information separately from the client application program makes the client application program portable and more flexible.

1. Writing the client application program Write the J2EE application client program on any development machine. At this stage, you do not require access to the WebSphere Application Server.

Rules: If you are writing a client application program that will run on z/OS, the following rules apply:

- Client programs may start their own transactions but cannot join in or start transactions in the WebSphere Application Server for z/OS run-time.
- Application client code must contain a main method.
- All input and output files for the application client must be in ASCII, because the client run-time runs in an ASCII JVM.

Using the `javax.naming.InitialContext` class, the client application program uses the lookup operation to access the Java Naming and Directory Interface (JNDI) name space. The `InitialContext` class provides the lookup method to locate resources.

The following example illustrates how a client application program uses the `InitialContext` class:

```
import javax.naming.*

public class MyAppClient
{
    public static void main(String argv[])
    {
        InitialContext initCtx = new InitialContext();
        Object homeObject =
            initCtx.lookup("java:comp/env/ejb/BasicCalculator");
        BasicCalculatorHome bcHome = (BasicCalculatorHome)
            javax.rmi.PortableRemoteObject.narrow(homeObject,
            BasicCalculatorHome.class);
        BasicCalculatorHome bc = bcHome.create();
        ...
    }
}
```

In this example, the program looks up an enterprise bean called `BasicCalculator`. The `BasicCalculator` EJB reference is located in the client JNDI name space at `java:comp/env/ejb/BasicCalculator`. Since the actual enterprise bean runs on the server, the application client run time returns a reference to the `BasicCalculator` home interface.

If the client application program lookup was for a resource reference or an environment entry, then lookup returns an instance of the configured type as defined by the client application deployment descriptor. For example, if the program lookup was a JDBC datasource, the lookup would return an instance of `javax.sql.DataSource`. Although you can edit deployment descriptor files, do not use the administrative console to modify them.

2. Assemble the application client using the Application Assembly Tool

The JNDI name space knows what to return on a lookup because of the information assembled by the Application Assembly Tool (AAT).

Assemble the J2EE application client on any development machine with the AAT installed.

When you use the Application Assembly Tool to assemble your application client, you provide the *application client* run time with the required information to initialize the execution environment for your client application program. Refer to the Application Assembly Tool description for implementation details.

Following is a list of things to keep in mind when you configure resources used by your client application program:

- When configuring resource references, resource environment references, and EJB references in the Application Assembly Tool, the General tab contains a required Name field. This field specifies where the application client run time binds the reference to the real object in the java:comp/env portion of the JNDI name space. The application client run time always binds these references relative to java:comp/env. For the programming example above, specify `ejb/BasicCalculator` in the Name field on the General tab of the Application Assembly Tool, which requires the program to perform a lookup of `java:comp/env/ejb/BasicCalculator`. If the Name field is set to `myString`, the resulting lookup is `java:comp/env/myString`.
- When configuring Resource references in the Application Assembly Tool, the Name field on the General tab is used for:
 - Binding a reference of that object type into the JNDI name space.
 - Retrieving client specific resource configuration information that was configured using the Application Client Resource Configuration Tool(ACRCT) on Windows, or the ACRCT scripting tool on z/OS.
- When configuring a resource reference in the Application Assembly Tool, the value in the Name field on the General tab must match the value in the JNDI Name field on the General tab for the resource in the Application Client Resource Configuration Tool.
- When configuring URL resources using the Application Client Resource Configuration Tool, the URL provider panel enables you to specify a protocol and a class to handle that protocol. If you want to use the default protocols, such as HTTP, you can leave those fields blank.
- When configuring resource references using the Application Assembly Tool, the General tab contains a field called Authorization. You can set this field to either Container or Application. If you set the field to Container, then the application client run time uses authorization information configured in the Application Client Resource Configuration tool for the resource. If the field is set to Application, then the application client run time expects the user application to provide authorization information for the resource. The application client run time ignores any authorization information configured with the Application Client Resource Configuration tool for that resource.
- When configuring resource environment references using the Application Assembly Tool, you must specify the location of the actual object in the server JNDI namespace using the Binding tab. A resource environment reference maps a logical name (the Name field on the general tab) used by the client application to the physical name of an object (the JNDI Name field on the Bindings tab). Not all objects bound into the server JNDI namespace are intended for use by an application client. For example, the WebSphere Application Server client run-time does not support the use of Java 2 Connector (J2C) objects on the client. The object needs to be remotable, and the client-side implementations must be made available on the application client run-time classpath.

- Resource environment references are different than resource references. Resource environment references allow your application client to use a logical name to look-up a resource bound into the server JNDI namespace. A resource reference allows your application to use a logical name to look-up a local J2EE resource. The J2EE specification does not specify a particular implementation of a resource. The following is a table of the supported resource types and identifies the resources to which the WebSphere Application Server provides a client implementation.

Resource Type	Client Configuration Notes	Client implementation provided by WebSphere Application Server
javax.sql.DataSource	Supports specification of any Datasource implementation class	No
java.net.URL	Supports specification of custom protocol handlers	Provided by Java Runtime Environment files
javax.mail.Session	Supports custom protocol configuration	Yes - POP3, SMTP, IMAP
javax.jms.QueueConnectionFactory, javax.jms.TopicConnectionFactory, javax.jms.Queue, javax.jms.Topic	Supports configuration of WebSphere Embedded Messaging, IBM MQ Series and other JMS providers	Yes - WebSphere Embedded Messaging

3. Assembling the Enterprise Archive (EAR)

The application is contained in an enterprise archive or .ear file. The .ear file is composed of:

- Enterprise bean, application client, and user-defined modules or .jar files
- Web applications or .war files
- Metadata describing the applications or application .xml files

You must assemble the .ear file on the server machine.

4. Distribute the EAR file

The client machines configured to run this client must have access to the .ear file.

If all the machines in your environment share the same image and platform, run the Application Client Resource Configuration Tool (ACRCT) on one machine to configure the external resources, and then distribute the configured .ear file to the other machines.

If your environment is set up with a variety of client installations and platforms, run the ACRCT for each unique configuration.

You can either distribute the .ear files to the correct client machines, or make them available on a network drive.

Distributing the .ear files is the responsibility of the system and network administrator.

5. Deploy the application client.

6. Configure the application client resources

If the client application defines the local resources, run the ACRCT (clientConfig command) on the local machine to reconfigure the .ear file. Use the ACRCT to change the configuration. For example, the .ear file can contain a DB2 resource, configured as C:\DB2. If, however, you installed DB2 in the D:\Program Files\DB2 directory, use the ACRCT to create a local version of the .ear file.

7. Deploy the application client.

If you plan to deploy the client on z/OS or OS/390, you have two options for running the Application Client Resource Configuration Tool (ACRCT):

- Run the ACRCT on Windows, or
- Run the ACRCT scripting tool on z/OS.

Both of these options produce equivalent output; only the tool interfaces are different. The ACRCT on Windows presents a graphical user interface, whereas the ACRCT for z/OS uses a scripting interface.

8. Use the WebSphere Administrative console to install the application client on z/OS or OS/390.

After developing the J2EE application client code, launch the application client. After completing these steps, launch the application client.

J2EE application client class loading

When you run your J2EE application client, a hierarchy of class loaders is created to load classes used by your application.

The following list describes the hierarchy of class loaders:

- The topmost class loader, the *bootstrap class loader*, contains the JAR files that make up the Java Virtual Machine code, such as `rt.jar`, plus those JAR files defined by the `-Xbootclasspath` parameter on the Java command. The WebSphere Application client run time sets this value to the `WAS_BOOTCLASSPATH` environment variable.
- The *extensions class loader* class loader is a child to the bootstrap class loader. This class loader contains JAR files in the `java/jre/lib/ext` directory or those JAR files defined by the `-Djava.ext.dirs` parameter on the Java command. The WebSphere Application Client run time does not set `-Djava.ext.dirs` parameters, so it uses the JAR files in the `java/jre/lib/ext` directory.
- The *system class loader* class loader contains JAR files and classes that are defined by the `-classpath` parameter on the Java command. The Application Client run time sets this parameter to the `WAS_CLASSPATH` environment variable.
- The *WebSphere class loader* class loader loads the WebSphere Application Client run time and any classes placed in the WebSphere Application Client user directories. The directories used by this class loader are defined by the `WAS_EXT_DIRS` environment variable. The `WAS_BOOTCLASSPATH`, `WAS_CLASSPATH`, and the `WAS_EXT_DIRS` environment variables are set in the `installation_root/bin/setupCmdLine` command shell for WebSphere Application Server server installations, or in the `installation_root/bin/setupClient` command shell for client installations.

As the J2EE application client run time initializes, additional class loaders are created as children of the WebSphere class loader. If your client application uses resources such as Java Database Connectivity (JDBC) API, Java Message Service (JMS) API, or Uniform Resource Locator (URL), a different class loader is created to load each of those resources. Finally, the application client run time sets the WebSphere class loader to load classes within the `.ear` file by processing the client JAR manifest repeatedly. The system classpath, defined by the `CLASSPATH` environment variable is never used and is not part of the hierarchy of class loaders.

To package your client application correctly, you must understand which class loader loads your classes. When Java loads a class, the class loader used to load that class is assigned to it. Any classes subsequently loaded by that class will use that class loader or any of its parents, but it will not use children class loaders.

In some cases the WebSphere Application Client run time can detect when your client application class is loaded by a different class loader from the one created for it by the WebSphere Application Client run time. When this detection occurs, you see the following message:

```
WSCL0205W: The incorrect class loader was used to load [0]
```

This message occurs when your client application class is loaded by one of the parent class loaders in the hierarchy. This situation is typically caused by having the same classes in the .ear file and on the hard drive. If one of the parent class loaders locates a class, that class loader loads it before the application client run time class loader. In some cases, your client application will still function correctly. In most cases, however, you receive "class not found" exceptions.

Configuring the classpath fields

When packaging your J2EE client application, you must configure various classpath fields. Ideally, you should package everything required by your application into your .ear file. This is the easiest way to distribute your J2EE client application to your clients. However, you should not package such resources as JDBC APIs, JMS APIs, or URLs. In the case of these resources, use classpath references to access those classes on the hard drive. You might also have other classes installed on your client machines that you do not need to redistribute. In this case, you also want to use classpath references to access the classes on the hard drive, as described below.

Referencing classes within the EAR file

WebSphere J2EE applications do not use the system class path. Use the MANIFEST Classpath entry to refer to other JARs within the .ear file. Configure these values using the module Classpath fields in the Application Assembly Tool. For example, if your client application needs to access the path of the enterprise bean JAR, add the deployed enterprise bean module name to your application client Classpath field in the Application Assembly Tool. The format of the Classpath field for each of the different modules (Application Client, enterprise bean, Web) is the same:

- The values must refer to .jar and .class files that are contained within the .ear file.
- The values must be relative to the root of the .ear file.
- The values cannot refer to absolute paths in the file systems.
- Multiple values must be separated by spaces, not colons or semi-colons.

Note: This is the Java method for allowing applications to function platform-independent.

Typically, you add modules (.jar files) to the root of the .ear file. In this case, you only need to specify the name of the module (.jar file) in the Classpath field. If you choose to add a module with a path, you need to specify the path relative to the root of the .ear file.

For referencing .class files, you must specify the directory relative to the root of the .ear file. With the Application Assembly Tool you can add individual class files to the .ear file. It is recommended that these additional class files are packaged in a .jar file. Add this .jar file to the module Classpath fields. If you add .class files to the root of the .ear file, add ./ to the module Classpath fields. Consider the following example directory structure in which the file myapp.ear contains an application client JAR file named client.jar and a mybeans.jar EJB module. Additional classes reside in class1.jar and utility/class2.zip files. A class

named xyz.class is not packaged in a JAR file but is in the root of the EAR file. Specify `./ mybeans.jar utility/class2.zip class1.jar` as the value of the Classpath property. The search order is: myapp.ear/client.jar myapp.ear/xyz.class myapp.ear/mybeans.jar myapp.ear/utility/class2.zip myapp.ear/class1.jar

Referencing classes that are not in the EAR file

Use the launchClient -CCclasspath parameter. This parameter is specified at run time and takes platform-specific classpath values, which means multiple values are separated by semi-colons or colons. There are many similarities between the client and the server in this respect.

Resource classpaths

When you configure resources used by your client application using the Application Client Resource Configuration Tool(ACRCT), or the z/OS ACRCT scripting tool, you can specify classpaths that are required by the resource. For example, if your application is using a JDBC to a DB2 database, add db2java.zip to the classpath field of the database provider. These classpath values are platform-specific and require semi-colons or colons to separate multiple values.

Using the launchClient API

If you use the **launchClient** shell and bat command, the WebSphere class loader hierarchy is created for you. However, if you use the launchClient API, you must perform this setup yourself. You should mimic the launchClient shell command in defining the Java system properties.

Developing pluggable application client code

As you prepare to install the pluggable application client, remember that pluggable clients are only available on Windows systems.

1. Install the pluggable application client from the WebSphere Application Client CD by selecting option **Pluggable Application Client** from the **Custom client installation** panel.
2. Set the Java application pluggable client environment by using the **setupClient** shell, located in:
install_root\AppClient\bin\setupClient.bat
3. Add your specific Java client application JAR files to the CLASSPATH and start your Java client application from this environment, after setting the environment variables.
4. Run the following Java command to invoke your client application (shown here on multiple lines for publication):

```
%JAVA_HOME%/bin/java -Xbootclasspath/p:%WAS_BOOTCLASSPATH%
-classpath <list of your application jars and classes>
-Djava.ext.dirs=%WAS_EXT_DIRS%
-Djava.naming.provider.url=iiop://<your WebSphere server machine name>
-Djava.naming.factory.initial=
com.ibm.websphere.naming.WsnInitialContextFactory %SERVER_ROOT% %CLIENTSAS%
<fully qualified class name to run>

$JAVA_HOME/bin/java -Xbootclasspath/p:$WAS_BOOTCLASSPATH
-classpath <list of your application jars and classes>
-Djava.ext.dirs=$WAS_EXT_DIRS
```

```
-Djava.naming.provider.url=iiop://<your WebSphere server machine name>
-Djava.naming.factory.initial=
com.ibm.websphere.naming.WsnInitialContextFactory $SERVER_ROOT $CLIENTSAS
<fully qualified class name to run>
```

View the Samples gallery for more information about application clients. Before you run the basicCalculator Sample, ensure the JMS Server is started.

Assembling Application Client Modules

If you want to use existing J2EE 1.2 Web modules in your J2EE 1.3 application, migrate them to J2EE 1.3 first.

Note: This task only applies to J2EE application clients.

Assemble a client module to contain application client code. (Group enterprise beans, Web components, and resource adapter code in separate modules).

1. Start the AAT.
2. From the New tab, select **Application Client**. Click **OK**. The navigation tree now displays various sets of properties for configuring the new application client.
3. Use the property dialog shown in the AAT workspace to change the default file name and location.
 - a. It is recommended that you change the display name so that it differs from the file name.
 - b. If you like, change the temporary location of the application client from the default location, `install_root/bin`.
4. Enter the main class filename and location.
 - a. Click **Browse** to locate the class file.
 - b. Select the archive containing the class files, and click **Select**.
 - c. Select the files you need from the archive, and click **OK**.
5. Define the assembly properties for the application client.
 - a. Right-click the assembly property in the navigation pane.
 - b. Select **New** in the right-click menu.
 - c. Enter values in the property dialog box.
 - d. Click **OK**.
6. Add files for the application client.
 - a. Right-click **Files** in the navigation pane.
 - b. Select **Add Files** from the right-click menu.
 - c. Locate the directory where the files are located and click **Select**.
 - d. Select the files to add and click **Add**.
 - e. In the **Selected files** window, click **OK**.
7. Save the application.

Assemble other new modules of your choice, if needed:

- Assembling EJB modules.
- Assembling resource adapter modules.

You can also migrate existing modules.

Another option is to proceed directly to assembling a new application module. While assembling an application module, you can create any new modules that you need.

After you finish assembling all of your application's modules, you are ready to deploy your application:

- on Windows, or
- on z/OS or OS/390.

Application client assembly settings

Use this page to specify assembly properties for J2EE application clients.

File name (Required, String)

Specifies the file name of the application client module, relative to the top level of the Enterprise Archive (EAR) file.

If this is a stand-alone module, the file name is the full path name of the archive.

Alternative DD

Specifies the file name for an alternative deployment descriptor file to use instead of the original deployment descriptor file in the module Java Archive (JAR) file. Although you can edit deployment descriptor files, do not use the administrative console to modify them.

This file is the post-assembly version of the deployment descriptor file. (Manually edit the original deployment descriptor file to resolve dependencies and security information. Directing the use of the alternative deployment descriptor allows you to keep the original deployment descriptor file intact). The value of the Alternative DD property must be the full path name of the deployment descriptor file relative to the module root directory. By convention, the file is in the ALT-INF directory. If this property is not specified, the deployment descriptor file is read directly from the module JAR file.

Classpath

Specifies the full classpath containing the dependent code that is not contained in the application client JAR file.

Specify the values relative to the root of the EAR file and separate the values with spaces. Absolute values that reference files or directories on the hard drive are ignored. To specify classes that are not in JAR files but are in the root of the EAR file, use a period and forward slash (.). Consider the following example directory structure in which the file myapp.ear contains an application client JAR file named client.jar. Additional classes reside in class1.jar and class2.zip files. A class named xyz.class is not packaged in a JAR file but is in the root of the EAR file myapp.ear/client.jar myapp.ear/class1.jar myapp.ear/class2.zip myapp.ear/xyz.class. Specify class1.jar class2.zip ./ as the value of the Classpath property. (Name only the directory for .class files.)

Display name (Required, String)

Specifies a short name that is intended for display by GUIs.

Small icon

Specifies a JPEG or GIF file containing a small image (16x16 pixels).

The image is used as an icon to represent the application client in a GUI.

Large icon

Specifies a JPEG or GIF file containing a large image (32x32 pixels).

The image is used as an icon to represent the application client in a GUI

Description

Contains text describing the application client.

Main class (Required, String)

Specifies the full path name of the main class for this application client.


```
property.my.resource.property.one=value1
property.my.resource.property.two=value2
```

3. On z/OS or OS/390, start the ACRCT scripting tool by invoking the shell script `clientConfig` in the UNIX System Services (USS) environment.

Example:

```
/usr/lpp/WebSphere/V5R0M0/bin/clientConfig.sh
```

Rule: You must specify the application client's Enterprise Archive (EAR) file on the command line. You may either specify resource parameters directly on the command line, or specify an input file. The syntax and parameter descriptions appear in Application Client Resource Configuration Tool (ACRCT) Scripting tool for z/OS.

- If the resource parameters are properly specified, the ACRCT scripting tool updates the application client's `client-resources.xmi` file with appropriate resource definitions.
- If the resource parameters are not properly specified or are missing, the ACRCT scripting tool issues an error message to the MVS console, and ends its processing.

Tip: If you receive an error message in response to the invocation, consider using the help function described in Application Client Resource Configuration Tool (ACRCT) Scripting tool for z/OS.

When the scripting tool successfully completes, the application client's EAR file is updated with the appropriate resource definitions.

When you have finished defining or updating the application client's resources, launch the application client.

Application Client Resource Configuration Scripting tool for z/OS

This section describes the command line syntax for the z/OS scripting version of the Application Client Resource Configuration Tool (ACRCT). The ACRCT scripting tool for z/OS allows you to:

- Define or delete resources for an application client that will run on z/OS.
- List the properties of a resource or provider that is already defined for the application client.

For define and delete actions, the ACRCT scripting tool alters the Enterprise Archive (EAR) file for the application client, as instructed by options you specify. If the ACRCT scripting tool encounters an error at any time during its processing, the tool issues an error message to the MVS console and terminates without changing the original contents of the application client EAR file.

When you use the ACRCT scripting tool, you may specify:

- More than one action (define, delete, or list) to perform for a specific application client.
- More than one EAR file, to define or delete resources for more than one application client at a time.

The command line invocation syntax for the ACRCT scripting tool follows. When you have a choice of one required keyword, those keywords appear within brackets [].

- To define or delete resources (shown here on multiple lines for publication):

```
acrct -earfile earfile [-define | -delete] [-provider | -resource]
[-f inputfile | key=value]
```

- To list resources:
acrct -list [-provider | -resource] [-f *inputfile* | -p *key=value*]
- To get help information:
acrct -help

Parameters

where:

-earfile

Is a required parameter that indicates the input filename of the application client EAR file.

earfilename

Identifies the location and name of the EAR file that contains the application client. This path and filename must directly follow the `-earfile` parameter.

-define

Instructs the scripting tool to define a provider or resource based on the input properties.

-delete

Instructs the scripting tool to delete a provider or resource based on the input properties.

-list

Instructs the scripting tool to the properties of a particular provider or resource, based on the input properties.

-help

Instructs the scripting tool to list basic examples and guidelines for using quotes around key values.

-provider

Indicates that the object to be defined or deleted, or for which properties are to be listed, is a provider.

-resource

Indicates that the object to be defined or deleted, or for which properties are to be listed, is a resource.

-f Indicates that the input properties for the provider or resource are provided in an input file, rather than specified directly on the command line.

inputfile

Identifies the location and name of the input file that contains the provider or resource properties. This path and filename must directly follow the `-f` parameter.

key=value

Specifies an input property for the provider or resource, in the form of key and value pairs.

Rules:

- You must use lowercase for keys.
- You cannot use blanks within a key and value pair; a blank signals the end of one key/value pair.
- Because blanks separate key and value pairs, you must be careful when a value you supply contains blanks. When you specify a value that contains blanks, enclose the value in single quotes or double quotes. Because some

shells processes quotes differently, you might have to do some testing to determine whether you must use single or double quotes.

Example: Suppose you invoke the scripting tool, passing this input (shown here on multiple lines for publication):

```
/WebSphere/V5R0M0/AppServer/bin:>acrct
-earfile usr/lpp/myapps/applclient2.ear -define
-provider providername='WebSphere JMS Provider'
```

The response is an error message along with an echo of the input string that the shell receives, followed by the input string as the scripting tool will process it (shown here on multiple lines for publication):

```
-earfile usr/lpp/myapps/applclient2.ear -define -provider providername=
WebSphere JMS Provider
```

String to be parsed by the Scripting Tool:

```
-earfile usr/lpp/myapps/applclient2.ear -define -provider providername=
WebSphere JMS Provider
```

```
Ear file is missing or is improperly specified.
Invalid syntax: -earfile usr/lpp/myapps/applclient2.ear -define -provider
providername=WebSphere JMS Provider
```

As you can see from the response, the shell has stripped off the single quotes, and passes invalid input to the scripting tool. To correct the problem, you need to use double quotes.

- The number of key and value pairs you specify depends on the type of resource or provider you are configuring. For each resource to be configured, use this information to determine which resource or provider properties are required.

The following examples demonstrate correct syntax:

Defining a new provider for an application client, using an input file:

```
acrct -earfile usr/lpp/myapps/applclient1.ear -define -provider -f
usr/lpp/myapps/inputProvider1.def
```

Defining a new provider for an application client, specifying properties directly on the command line:

```
acrct -earfile usr/lpp/myapps/applclient2.ear -define -provider -p
providertype=DataSourceProvider name=DB2UDBV7
```

Defining a new provider and deleting the resource it replaces, in the same EAR file:

```
acrct -earfile usr/lpp/myapps/applclient1.ear -define -provider -f
usr/lpp/myapps/inputProvider2.def -delete -resource -f
usr/lpp/myapps/inputProvider1.def
```

Defining a new provider in more than one EAR file:

```
acrct -earfile usr/lpp/myapps/applclient1.ear -define -provider -f
usr/lpp/myapps/inputProvider2.def -earfile
usr/lpp/myapps/applclient2.ear -define -provider -f
usr/lpp/myapps/inputProvider2.def
```

Determining required properties for z/OS application client resources

When you deploy application clients on z/OS or OS/390, you need to determine the required properties to specify when using the z/OS scripting version of the Application Client Resource Configuration Tool (ACRCT).

Note: This procedure applies only for J2EE application clients.

Use the following information to determine required properties to specify for application client resources.

If the application uses this type of resource or provider:	Find required and optional properties in these articles:
Data source	<ul style="list-style-type: none"> • Datasource Provider • Datasource
JMS	<ul style="list-style-type: none"> • JMS Provider • JMS Connection • JMS Destination
Mail session	<ul style="list-style-type: none"> • Mail provider • Mail session
Resource environment	<ul style="list-style-type: none"> • Resource environment provider • Resource environment entry
URL	<ul style="list-style-type: none"> • URL provider • URL factory
WebSphere MQ queue	<ul style="list-style-type: none"> • WebSphere MQ queue connection factory • WebSphere MQ queue destination factory
WebSphere MQ topic	<ul style="list-style-type: none"> • WebSphere MQ topic connection factory • WebSphere MQ topic destination factory
WebSphere queue	<ul style="list-style-type: none"> • WebSphere queue connection factory • WebSphere queue destination factory
WebSphere topic	<ul style="list-style-type: none"> • WebSphere topic connection factory • WebSphere topic destination factory

Properties for data source providers

name	required
description	optional
implementation	optional
classpath	optional

Properties for data sources

providertype=DataSourceProvider	required
providertype=DataSourceProvider	required
name	required
description	optional
jndiname	required
databasename	optional
user	optional
password	optional

```

providertype=DataSourceProvider
providertype=DataSourceProvider
name="PolicyDatasource"
description="Datasource for Policy App"
jndiname=jdbc/PolicyDS

```

databasename=POLICYAPP
 user=dbuser
 password=dbpw

Properties for JMS providers

name	required
description	optional
classpath	optional
externalinitialcontextfactory	optional
externalproviderurl	optional

Properties for JMS connections

providertype=JMSPProvider	required	
providertype	required	
name	required	
description	optional	
jndiname	required	
externaljndiname	optional	
type	required	Valid values are: • QUEUE • TOPIC
user	optional	
password	optional	

Properties for JMS destinations

providertype=JMSPProvider	required	
providertype	required	
name	required	
description	optional	
jndiname	required	
externaljndiname	optional	
type	required	Valid values are: • QUEUE • TOPIC

Properties for mail providers

name	required
description	optional
classpath	optional

Properties for mail sessions

providertype=MailProvider	required	
---------------------------	----------	--

providername	required	
name	required	
description	optional	
jndiname	required	
mailfrom	optional	
mailstorehost	optional	
mailstoreuser	optional	
mailstorepassword	optional	
mailtransporthost	optional	
mailtransportprotocol	required	Valid values are: • smtp • imap • pop3
mailtransportuser	optional	
mailtransportpassword	optional	
debug	required	Valid values are: • True • False

Properties for resource environment providers

name	required
description	optional
classpath	optional

Properties for resource environment entries

providertype=ResourceEnvironmentProvider	required
providername	required
name	required
description	optional
jndiname	required

Properties for URL providers

name	required
description	optional
protocol	optional
classpath	optional
streamhandlerclass	optional

Properties for URL factories

providertype=URLProvider	required
providername	required

name	required
description	optional
jndiname	required
url	required

Properties for WebSphere MQ queue connection factories

providertype=JMSProvider	required	
providertype="MQ JMS Provider"	required	
name	required	
description	optional	
jndiname	required	
transporttype	required	Valid values are: • CLIENT • BINDINGS
clientid	optional	
user	optional	
password	optional	
channel	optional	
ccsid	optional	

Properties for WebSphere MQ queue destination factories

providertype=JMSProvider	required	
providertype="MQ JMS Provider"	required	
name	required	
description	optional	
jndiname	required	
persistence	required	Valid values are: • APPLICATION_DEFINED • QUEUE_DEFINED • PERSISTENT • NONPERSISTENT
priority	required	Valid values are: • APPLICATION_DEFINED • QUEUE_DEFINED • <i>specified_integer</i> from 0 through 9
expiry	required	Valid values are: • APPLICATION_DEFINED • UNLIMITED • <i>specified_value</i>
basequeueName	required	
basequeueManagerName	optional	

targetclient	required	Valid values are: • JMS • MQ
ccsid	optional	
usenativeencoding	required	Valid values are: • True • False

Properties for WebSphere MQ topic connection factories

providertype=JMSProvider	required	
providertype="MQ JMS Provider"	required	
name	required	
description	optional	
jndiname	required	
transporttype	required	Valid values are: • CLIENT • BINDINGS
clientid	optional	
brokercontrolqueue	optional	
brokerqueuemanager	optional	
brokerpubqueue	optional	
brokersubqueue	optional	
brokerccsubq	optional	
brokerversion	required	Valid values are: • MAOC • MQSI
userid	optional	
password	optional	
ccsid	optional	
channel	optional	

Properties for WebSphere MQ topic destination factories

providertype=JMSProvider	required	
providertype="MQ JMS Provider"	required	
name	required	
description	optional	
jndiname	required	
persistence	required	Valid values are: • APPLICATION_DEFINED • QUEUE_DEFINED • PERSISTENT • NONPERSISTENT

priority	required	Valid values are: <ul style="list-style-type: none"> APPLICATION_DEFINED QUEUE_DEFINED <i>specified_integer</i> from 0 through 9
expiry	required	Valid values are: <ul style="list-style-type: none"> APPLICATION_DEFINED UNLIMITED <i>specified_value</i>
basetopicname	required	
targetclient	required	Valid values are: <ul style="list-style-type: none"> JMS MQ
brokerdursubqueue	optional	
brokerccdursubqueue	optional	
ccsid	optional	
usenativeencoding	required	Valid values are: <ul style="list-style-type: none"> True False

Properties for WebSphere queue connection factories

providertype=JMSPProvider	required
providertype="WebSphere JMS Provider"	required
name	required
description	optional
jndiname	required
node	required
servername	required
user	optional
password	optional

Properties for WebSphere queue destination factories

providertype=JMSPProvider	required	
providertype="WebSphere JMS Provider"	required	
name	required	
description	optional	
jndiname	required	
node	required	
persistence	required	Valid values are: <ul style="list-style-type: none"> APPLICATION_DEFINED PERSISTENT NONPERSISTENT

priority	required	Valid values are: <ul style="list-style-type: none"> • APPLICATION_DEFINED • <i>specified_integer</i> from 0 through 9
expiry	required	Valid values are: <ul style="list-style-type: none"> • APPLICATION_DEFINED • UNLIMITED • <i>specified_value</i>

Properties for WebSphere topic connection factories

providertype=JMSPProvider	required	
providertype="WebSphere JMS Provider"	required	
name	required	
description	optional	
jndiname	required	
servername	required	
node	required	
port	required	Valid values are: <ul style="list-style-type: none"> • QUEUED • DIRECT
clientid	optional	
userid	optional	
password	optional	

Properties for WebSphere topic destination factories

providertype=JMSPProvider	required	
providertype="WebSphere JMS Provider"	required	
name	required	
description	optional	
jndiname	required	
topic	required	
persistence	required	Valid values are: <ul style="list-style-type: none"> • APPLICATION_DEFINED • PERSISTENT • NONPERSISTENT
priority	required	Valid values are: <ul style="list-style-type: none"> • APPLICATION_DEFINED • <i>specified_integer</i> from 0 through 9
expiry	required	Valid values are: <ul style="list-style-type: none"> • APPLICATION_DEFINED • UNLIMITED • <i>specified_value</i>

Deploying application clients on workstation platforms

After developing an application client, deploy this application on client machines. *Deployment* consists of pulling together the various artifacts that the application client requires.

The *Application Client Resource Configuration Tool* (ACRCT) defines resources for the application client. These configurations are stored in the application client .ear file. The application client run time uses these configurations for resolving and creating an instance of the resources for the application client.

If you plan to deploy the client on z/OS or OS/390, you have two options:

- Run the Application Client Resource Configuration Tool (ACRCT) on Windows, or
- Run the Client Container Resource Configuration Scripting tool on z/OS.

Both of these tools produce equivalent output. They both provide resource definitions for an application client, which are stored in the application client .ear file. The application client run time (or container) uses these configurations for resolving and creating an instance of the resources for the application client.

Note: This task only applies to J2EE application clients. Only perform this task if you configured your J2EE application client to use resource references.

1. Start the ACRCT and open an EAR file.
2. Configure new data source providers.
3. Configure mail providers and sessions.
4. Configure URL providers and sessions.
5. Configure Java messaging client resources.
6. Configure new environment entries.
7. (Optional) Remove application client resources.
8. Save the EAR file.

Starting the Application Client Resource Configuration Tool and opening an EAR file

Note: This task only applies to J2EE application clients.

1. Open a command prompt and change to the `install_root\bin` directory.
2. Run the `clientConfig.bat` file for a Windows system or the `clientConfig.sh` file for a UNIX system.
3. Open an EAR file within the Application Client Resource Configuration Tool (ACRCT):
 - Click **File > Open**.
 - Select the file and click **Open**.
4. Save your changes to the file and close the tool:
 - Click **File > Save**.
 - Click **File > Exit**.

Data sources for application clients

The J2EE application client does not support looking up or directly accessing data sources configured on WebSphere Application Server because the J2EE application client does not support Java 2 Connection Factories. To use a data source directly

from the client application, you must use the ACRCT to configure your data source. In addition, WebSphere Application Server and WebSphere Application Server clients do not provide client database drivers to be used directly from a J2EE application client. If your application client accesses a database directly, you must provide the database drivers on the client machine. You might contact your database vendor to acquire client database driver code and licenses. Instead of accessing the database directly, it is recommended that your client application use an enterprise bean. Accessing a database through an enterprise bean eliminates the need to have database drivers on the client machine, since the database access is handled by the enterprise bean running on the WebSphere Application Server. For a current list of providers that are supported on the WebSphere Application Server go the following site:

Supported hardware, software, and APIs

Configuring new data source providers (JDBC providers) for application clients

During this task, you create new data source providers, also known as JDBC providers, for your application client. In a separate administrative task, install the Java code for the required data source provider on the client machine on which the application client resides.

1. Start the tool and open the EAR file for which you want to configure the new data source provider. The EAR file contents display in a tree view.
2. Select the JAR file in which you want to configure the new data source provider from the tree.
3. Expand the JAR file to view its contents.
4. Click the **Data Source Providers** folder. Do one of the following:
 - Right-click the folder and click **New Provider**.
 - Click **Edit > New** on the menu bar.
5. Configure the data source provider properties in the resulting property dialog.
6. Click **OK** when you finish.
7. Click **File > Save** on the menu bar to save your changes.

Configuring new data source providers

During this task, you will create new data source providers, also known as JDBC drivers, for your application client. In a separate administrative task, install the Java code for the required data source provider on the client machine where the application client resides.

1. Start the ACRCT, click **File > Open**, and select the EAR file for which you want to configure the new data source provider. The EAR file contents display in a tree view.
2. Select the JAR file in which you want to configure the new data source provider from the tree.
3. Expand the JAR file to view its contents.
4. Right click the **Data Source Providers** folder and select **New Provider**.
5. Configure the data source provider properties in the resulting property dialog.
6. Click **OK**.
7. Click **File > Save** to save your changes.

Example: Configuring data source provider and data source settings: The purpose of this article is to help you to configure data source provider and data source settings.

- Required fields:
 - Data Source Provider Properties page: name
 - Data Source Properties page: name, jndiName
- Special cases:
 - The user name and password fields have no equivalent xmi tags. You must specify these fields in the custom properties.
 - The password is encrypted when you use the Application Client Resource Configuration Tool (ACRCT). If you do not use the ACRCT, the field cannot be encrypted.
- Example (shown here on multiple lines for publication):

```
<resources.jdbc:JDBCProvider xmi:id="JDBCProvider_1"
  name="jdbcProvider:name" description="jdbcProvider:description"
  implementationClassName="jdbcProvider:ImplementationClass">
<classpath>jdbcProvider:classpath</classpath>
<factories xmi:type="resources.jdbc:WAS40DataSource"
  xmi:id="WAS40DataSource_1" name="jdbcFactory:name"
  jndiName="jdbcFactory:jndiName" description="jdbcFactory:description"
  databaseName="jdbcFactory:databaseName">
<propertySet xmi:id="J2EEResourcePropertySet_13">
<resourceProperties xmi:id="J2EEResourceProperty_13"
  name="jdbcFactory:customName" value="jdbcFactory:customValue"/>
<resourceProperties xmi:id="J2EEResourceProperty_14" name="user"
  value="jdbcFactory:user"/>
<resourceProperties xmi:id="J2EEResourceProperty_15" name="password"
  value="{xor}NTs9PBk+PCswLSZ1MT4y0g==" />
</propertySet>
</factories>
<propertySet xmi:id="J2EEResourcePropertySet_14">
<resourceProperties xmi:id="J2EEResourceProperty_16"
  name="jdbcProvider:customName" value="jdbcProvider:customValue"/>
</propertySet>
</resources.jdbc:JDBCProvider>
```

Data source provider settings for application clients:

Use this page to create a data source under a JDBC provider which provides the specific JDBC driver implementation class.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File > Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file. Right-click **Data Source Providers >** and click **New**. The following fields appear on the **General** tab:

Name:

Specifies the display name for the data source.

For example you can set this field to *Test Data Source*.

Data type String

Description:

Specifies a text description for the resource.

Data type String

Class Path:

A list of paths or jarfile names which together form the location for the resource provider classes.

Implementation class:

Use this setting to perform database specific functions.

Data type	String
Default	Dependent on JDBC driver implementation class

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at runtime for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Data source properties for application clients:

Use this page to create or modify the data sources.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File > Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Data Source Providers > Data source provider instance**. Right-click **Data Sources** and click **New**. The following fields are displayed on the **General** tab:

Name:

Specifies the display name of this data source.

Data type	String
------------------	--------

Description:

Specifies a text description of the data source.

Data type	String
------------------	--------

JNDI Name:

The application client run time uses this field to retrieve configuration information. The name must match the value of the **Name** field on the General tab in the Application Client Resource Reference section of the Assembly Tool.

Database Name:

The name of the database to which you want to connect.

User:

Use the user ID with the Password property, for authentication if the calling application does not provide a userid and password explicitly.

If you specify a value for the User ID property, you must also specify a value for the Password property. The connection factory User ID and Password properties are used if the calling application does not provide a userid and password explicitly.

Password:

Use the password with the User ID property, for authentication if the calling application does not provide a userid and password explicitly.

If you specify a value for the User ID property, you must also specify a value for the Password property.

Re-Enter Password:

Confirms the password.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at runtime for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Configuring new data sources for application clients

During this task, you create new data sources for your application client.

1. Click the data source provider for which you want to create a data source in the tree. Do one of the following:
 - Configure a new data source provider.
 - Click an existing data source provider.
2. Expand the data source provider to view its **Data Sources** folder.
3. Click the folder. Do one of the following:
 - Right-click the folder and click **New Factory**.
 - Click **Edit > New** on the menu bar.
4. Configure the data source properties in the resulting property dialog.
5. Click **OK** when you finish.
6. Click **File > Save** on the menu bar to save your changes.

Configuring mail providers and sessions for application clients

Use the Application Client Resource Configuration Tool (ACRCT) to edit the configurations of JavaMail sessions and providers for your application clients to use.

1. Open the ACRCT.
2. Open an EAR file.
3. Locate the JavaMail objects in the tree that displays. For example, if your file contains JavaMail sessions, expand **Resources** > *application.jar* > **JavaMail Providers** > *java_mail_provider_instance* > **JavaMail Sessions**.
In this example, *java_mail_provider_instance* is a particular JavaMail provider.

The JavaMail session instances are located in the **JavaMail Sessions** folder.

Mail provider settings for application clients

Use this page to implement the JavaMail API and create mail sessions.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file. Right-click **Mail Providers** > and click **New**. The following fields appear on the **General** tab:

Name:

The name of the JavaMail resource provider.

Description:

An optional description for the resource provider.

Class Path:

Specifies a list of paths or JAR file names which together form the location for the resource provider classes.

Protocol:

Specifies the name of the protocol.

Classname:

Specifies the name of the class implementing the protocol. Leave this field blank if you want to use the default implementation.

Type:

This menu contains the following two values: **TRANSPORT** or **STORE**.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Mail session settings for application clients

Use this page to configure mail session properties.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected

JAR file > **Mail Providers** > *mail provider instance*. Right-click **Mail Sessions** and click **New**. The following fields appear on the **General** tab:

Name:

Represents the administrative name of the JavaMail session object.

Description:

Provides an optional description for your administrative records.

JNDI Name:

The application client run-time uses this field to retrieve configuration information. The name must match the value of the **Name** field on the General tab in the Application Client Resource Reference section of the Application Assembly Tool.

Mail Transport Host:

Specifies the server to connect to when sending mail.

Mail Transport Protocol:

Specifies the transport protocol to use when sending mail.

Mail Transport User:

Specifies the user ID to use when the mail transport host requires authentication.

Mail Transport Password:

Specifies the password to use when the mail transport host requires authentication.

Re-Enter Password:

Confirms the password.

Mail From:

Specifies the mail originator.

Mail Store Host:

Mail account host (or "domain") name.

Mail Store User:

The user ID of the mail account.

Mail Store Password:

The password of the mail account.

Re-Enter Password:

Confirms the password.

Mail Store Protocol:

Specifies the protocol to be used when receiving mail.

Mail Debug:

When true, JavaMail interaction with mail servers, along with these mail session properties will be printed to stdout.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at runtime for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Example: Configuring JavaMail provider and JavaMail session settings for application clients

The purpose of this article is to help you configure JavaMail provider and JavaMail session settings.

- Required fields:
 - JavaMail Provider Properties page: name, and at least one protocol provider
 - JavaMail Session Properties page: name, jndiName, mail transport protocol, mail store protocol
- Special cases:
 - The password is encrypted when using the ACRCT tool. Without the tool, you cannot encrypt this field.
- Example (shown here on multiple lines for publication):

```
<resources.mail:MailProvider xmi:id="MailProvider_1"
  name="Default Mail Provider" description="IBM JavaMail Implementation">
<classpath>mailProvider:classpath</classpath>
<factories xmi:type="resources.mail:MailSession" xmi:id="MailSession_1"
  name="mailSession:name" jndiName="mailSession:jndiName"
  description="mailSession:description"
  mailTransportHost="mailSession:mailTransportHost"
  mailTransportUser="mailSession:mailTransportUser"
  mailTransportPassword="{xor}Mj42Mww6LCw2MDF1MT4y0g=="
  mailFrom="mailSession:mailFrom" mailStoreHost="mailSession:mailStoreHost"
  mailStoreUser="mailSession:mailStoreUser"
  mailStorePassword="{xor}Mj42Mww6LCw2MDF1MT4y0g==" debug="true"
  mailTransportProtocol="ProtocolProvider_1"
  mailStoreProvider="ProtocolProvider_1">
<propertySet xmi:id="J2EEResourcePropertySet_1">
<resourceProperties xmi:id="J2EEResourceProperty_1"
  name="mailSession:customName" value="mailSession:customValue"/>
</propertySet>
</factories>
<propertySet xmi:id="J2EEResourcePropertySet_2">
<resourceProperties xmi:id="J2EEResourceProperty_2"
  name="mailProvider:customName" value="mailProvider:customValue"/>
</propertySet>
<protocolProviders xmi:id="ProtocolProvider_1" protocol="smtp"
  classname="smtp:className"/>
<protocolProviders xmi:id="ProtocolProvider_2" protocol="pop3"
  classname="pop3:className"/>
<protocolProviders xmi:id="ProtocolProvider_3" protocol="imap"
  classname="imap:className"/>
</resources.mail:MailProvider>
```

Configuring new mail sessions for application clients

During this task, you configure new mail sessions for your application client. The mail sessions are associated with the pre-configured default mail provider supplied by the product.

1. Start the Application Client Resource Configuration Tool (ACRCT) and open the EAR file. The EAR file contents are displayed in a tree view.
2. Select the JAR file in which you want to configure the new JavaMail session.
3. Expand the JAR file to view its contents.
4. Click **JavaMail Providers > MailProvider > JavaMail Sessions**. Complete one of the following actions:
 - Right-click the **JavaMail Sessions** folder and select **New Factory**.
 - Click **Edit > New** on the menu bar.
5. Configure the JavaMail session properties in the resulting property dialog.
6. Click **OK**.
7. Click **File > Save** on the menu bar to save your changes.

URLs for application clients

A *Uniform Resource Locator* (URL) is an identifier that points to an electronically accessible resource, such as a directory file on a machine in a network, or a document stored in a database.

URLs appear in the format *scheme:scheme_information*.

You can represent a *scheme* as *http*, *ftp*, *file*, or another term that identifies the type of resource and the mechanism by which you can access the resource.

In a World Wide Web browser location or address box, a URL for a file available using HyperText Transfer Protocol (HTTP) starts with *http:*. An example is *http://www.ibm.com*. Files available using File Transfer Protocol (FTP) start with *ftp:*. Files available locally start with *file:*.

The *scheme_information* commonly identifies the Internet machine making a resource available, the path to that resource, and the resource name. The *scheme_information* for HTTP, FTP and File generally starts with two slashes (*//*), then provides the Internet address separated from the resource path name with one slash (*/*). For example,

http://www-4.ibm.com/software/webservers/appserv/library.html.

For HTTP and FTP, the path name ends in a slash when the URL points to a directory. In such cases, the server generally returns the default index for the directory.

URL providers for the Application Client Resource Configuration Tool

A URL provider implements the function for a particular URL protocol, such as Hyper Text Transfer Protocol (HTTP). This provider, comprised of a pair of classes, extends the *java.net.URLStreamHandler* and *java.net.URLConnection* classes.

Configuring new URL providers for application clients

During this task, you create URL providers and URLs for your client application. In a separate administrative task, you must install the Java code for the required URL provider on the client machine on which the client application resides.

1. Start the Application Client Resource Configuration Tool (ACRCT).
2. Open the EAR file for which you want to configure the new URL provider. The EAR file contents display in a tree view.
3. Select the JAR file in which you want to configure the new URL provider from the tree.
4. Expand the JAR file to view its contents.
5. Click the folder called **URL Providers**. Complete one of the following actions:
 - Right-click the folder and click **New Provider**.
 - Click **Edit > New** on the menu bar.
6. Configure the URL provider properties in the resulting property dialog.
7. Click **OK**.
8. Click **File > Save** on the menu bar to save your changes.

Configuring URL providers and sessions using the Application Client Resource Configuration Tool

Use the Application Client Resource Configuration Tool (ACRCT) to edit the configurations of URL providers and URLs to be used by your application clients.

1. Open the ACRCT.
2. Open an EAR file.
3. Locate the URL objects in the tree that displays. For example, if your file contains URL providers and URLs, expand **Resources** -> *application.jar* -> **URL Providers** -> *url_provider_instance* where *url_provider_instance* is a particular URL provider.
4. If you expand the tree further, you will also see the **URLs** folders containing the URL instances for each URL provider instance.

URL settings for application clients:

Use this page to implement the function for a particular URL protocol, such as Hyper Text Transfer Protocol (HTTP).

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File > Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **URL Providers** > *URL provider instance*. Right-click **URLs** and click **New**. The following fields appear on the **General** tab.

This provider, comprised of classes, extends the `java.net.URLStreamHandler` and `java.net.URLConnection` classes.

Name:

Administrative name for the URL

Description:

Optional description of the URL, for your administrative records

JNDI Name:

The application client run-time uses this field to retrieve configuration information. The name must match the value of the **Name** field on the General tab in the Application Client Resource Reference section of the Application Assembly Tool.

URL:

A Uniform Resource Locator (URL) name that points to an internet or intranet resource. For example: `http://www.ibm.com`

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at runtime for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

URL provider settings for application clients:

Use this page create new URLs.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File > Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file. Right-click **URL Providers >** and click **New**. The following fields appear on the **General** tab.

A URL provider implements the function for a particular URL protocol, such as Hyper Text Transfer Protocol (HTTP). This provider, comprised of classes, extends the `java.net.URLStreamHandler` and `java.net.URLConnection` classes.

Name:

Administrative name for the URL

Description:

Optional description of the URL, for your administrative records

Class Path:

A list of paths or JAR file names which together form the location for the resource provider classes.

Protocol:

Protocol supported by this stream handler. For example, "nntp", "smtp", "ftp", and so on.

To use the default protocol, leave this field blank.

Stream handler class:

Fully qualified name of a User-defined Java class that extends `java.net.URLStreamHandler` for a particular URL protocol, such as FTP.

To use the default stream handler, leave this field blank.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at runtime for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Example: Configuring URL and URL provider settings for application clients

The purpose of this article is to help you to configure URL and URL provider settings.

- Required fields:
 - URL Properties page: name, jndiName, url
 - URL Provider Properties page: name
- Example (shown here on multiple lines for publication):

```
<resources.url:URLProvider xmi:id="URLProvider_1" name="urlProvider:name"
  description="urlProvider:description"
  streamHandlerClassName="urlProvider:streamHandlerClass"
  protocol="urlProvider:protocol">
<classpath>urlProvider:classpath</classpath>
<factories xmi:type="resources.url:URL" xmi:id="URL_1" name="urlFactory:name"
  jndiName="urlFactory:jndiName" description="urlFactory:description"
  spec="urlFactory:url">
<propertySet xmi:id="J2EEResourcePropertySet_18">
<resourceProperties xmi:id="J2EEResourceProperty_20"
  name="urlFactory:customName" value="urlFactory:customValue"/>
</propertySet>
</factories>
<propertySet xmi:id="J2EEResourcePropertySet_19">
<resourceProperties xmi:id="J2EEResourceProperty_21"
  name="urlProvider:customName" value="urlProvider:customValue"/>
</propertySet>
</resources.url:URLProvider>
```

Configuring new URLs with the Application Client Resource Configuration Tool

During this task, you create URLs for your client application.

1. Click the URL provider for which you want to create a URL in the tree. Do one of the following:
 - Configure a new URL provider.
 - Click an existing URL provider.
2. Expand the URL provider to view the **URLs** folder.
3. Click the folder. Do one of the following:
 - Right-click the folder and click **New Factory**.
 - Click **Edit -> New** on the menu bar.
4. Configure the URL properties in the resulting property dialog.
5. Click **OK** when you finish.
6. Click **File -> Save** in the menu bar to save your changes.

WebSphere asynchronous messaging using the Java Message Service API for the Application Client Resource Configuration Tool

WebSphere Application Server supports asynchronous messaging as a method of communication based on the Java Message Service (JMS) programming interface. The JMS interface provides a common way for Java programs (clients and J2EE applications) to create, send, receive, and read asynchronous requests as JMS messages.

This topic provides an overview of asynchronous messaging using JMS support provided by the WebSphere Application Server.

The base support for asynchronous messaging using the JMS API provides the common set of JMS interfaces and associated semantics that define how a JMS client can access the facilities of a JMS provider. This support enables WebSphere J2EE applications, as JMS clients, to exchange messages asynchronously with other JMS clients, by using JMS destinations (queues or topics). A J2EE application can use JMS queue destinations for point-to-point messaging and JMS topic destinations for Pub and Sub messaging. A J2EE application can explicitly poll for messages on a destination then retrieve messages for processing by business logic beans (enterprise beans).

With the base JMS/XA support, the J2EE application uses standard JMS calls to process messages, including any responses or outbound messaging. An enterprise bean can handle responses acting as a sender bean, or within the enterprise bean that receives the incoming messages. Optionally, this process can use two-phase commit within the scope of a transaction. This level of function for asynchronous messaging is called *bean-managed messaging*, and gives an enterprise bean complete control over the messaging infrastructure; for example, connection and session pool management. The common container has no role in *bean-managed messaging*.

WebSphere Application Server also supports automatic asynchronous messaging using message-driven beans (a type of enterprise bean defined in the EJB 2.0 specification) and JMS listeners (part of the JMS application server facilities). Messages are automatically retrieved from JMS destinations, optionally within a transaction, then sent to the message-driven bean in a J2EE application, without the application having to explicitly poll JMS destinations.

Configuring Java messaging client resources

In a separate administrative task, install the Java Message Service (JMS) client on the client machine where the application client resides. The messaging product vendor must provide an implementation of the JMS client. For more information, see your messaging product documentation.

During this task, you create new JMS provider configurations for your application client. The application client can use a messaging service through the Java Message Service APIs. A JMS provider provides two kinds of J2EE factories. One is a *JMS connection factory*, and the other is a *JMS destination factory*.

1. Start the Application Client Resource Configuration Tool (ACRCT).
2. Open the EAR file for which you want to configure the new JMS provider. The EAR file contents are in the displayed tree view.
3. Select the JAR file in which you want to configure the new JMS provider from the tree.

4. Expand the JAR file to view its contents.
5. Click the **JMS Providers** folder and click **New Provider**.
6. Configure the JMS provider properties in the resulting property dialog.
7. Click **OK**.
8. Click **File > Save**.

Configuring new JMS providers with the Application Client Resource Configuration Tool

During this task, you will create new JMS provider configurations for your application client. The application client can make use of a messaging service through the Java Message Service APIs. A JMS provider provides two kinds of J2EE factories. One is a JMS Connection factory, and the other is a JMS destination factory.

In a separate administrative task, you must install the JMS client on the client machine where the application client resides. The messaging product vendor must provide an implementation of the JMS client. For more information, see your messaging product documentation.

1. Start the tool and open the EAR file for which you want to configure the new JMS provider. The EAR file contents will be displayed in a tree view.
2. From the tree, select the JAR file in which you want to configure the new JMS provider.
3. Expand the JAR file to view its contents.
4. Click the folder called **JMS Providers**. Do one of the following:
 - Right-click the folder and select **New Provider**.
 - On the menu bar, click **Edit -> New**.
5. In the resulting property dialog, configure the JMS provider properties.
6. When finished, click **OK**.
7. On the menu bar, click **File -> Save** to save your changes.

JMS provider settings for application clients

Use this page to configure properties of the JMS provider, if you want to use a JMS provider other than the internal WebSphere JMS provider or the MQSeries JMS provider.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File > Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file. Right-click **JMS Providers > click New**. The following fields appear on the **General** tab.

Name:

The name by which the JMS provider is known for administrative purposes.

Data type String

Description:

A description of the JMS provider, for administrative purposes

Data type String

Class Path:

A list of paths or jarfile names which together form the location for the resource provider classes.

Context factory class:

The Java classname of the initial context factory for the JMS provider.

For example, for an LDAP service provider the value has the form:
com.sun.jndi.ldap.LdapCtxFactory.

Data type String

Provider URL:

The JMS provider URL for external JNDI lookups.

For example, an LDAP URL for a JMS provider has the form:
ldap://hostname.company.com/contextName.

Data type String

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at runtime for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

WebSphere queue connection factory settings for application clients

Use this panel to view or change the configuration properties of the selected queue connection factory for use with the internal WebSphere JMS provider that is installed with WebSphere Application Server. These configuration properties control how connections are created to the associated JMS queue destination. To view this Application Client Resource Configuration Tool (ACRCT) page, click **File > Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **JMS Providers > JMS provider instance**. Right-click **WAS Queue Connection Factories** and click **New**. The following fields appear on the **General** tab.

A queue connection factory is used to create JMS connections to queue destinations. The queue connection factory is created by the internal WebSphere JMS provider. A queue connection factory for the internal WebSphere JMS provider has the following properties:

Name:

The name by which this queue connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the JMS connection factories across the WebSphere administrative domain.

Data type String

Description:

A description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type	String
Default	Null

JNDI Name:

The application client run-time uses this field to retrieve configuration information. The name must match the value of the **Name** field on the General tab in the Application Client Resource Reference section of the Application Assembly Tool.

User:

The User ID used, with the **Password** property, for authentication if the calling application does not provide a `userid` and password explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

The connection factory **User ID** and **Password** properties are used if the calling application does not provide a User ID and password explicitly; for example, if the calling application uses the method `createQueueConnection()`. The JMS client flows the `userid` and password to the JMS server.

Data type	String
------------------	--------

Password:

The password used, with the **User ID** property, for authentication if the calling application does not provide a `userid` and password explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

Re-Enter Password:

Confirms the password.

Node:

The WebSphere node name of the administrative node where the JMS server runs for this connection factory. Connections created by this factory connect to that JMS server.

Data type	String
------------------	--------

Application Server:

Enter the name of the application server. This name is not the host name of the machine, but the name of the configured application server.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

WebSphere topic connection factory settings for application clients

Use this panel to view or change the configuration properties of the selected topic connection factory for use with the internal WebSphere JMS provider. These configuration properties control how connections are created to the associated JMS topic destination.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File > Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **JMS Providers > JMS provider instance**. Right-click **WAS Topic Connection Factories** and click **New**. The following fields appear on the **General** tab.

A topic connection factory is used to create JMS connections to topic destinations. The topic connection factory is created by the associated JMS provider. A topic connection factory for the internal WebSphere JMS provider has the following properties.

Name:

The name by which this queue connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the JMS connection factories across the WebSphere administrative domain.

Data type	String
------------------	--------

Description:

A description of this topic connection factory for administrative purposes within IBM WebSphere Application Server.

Data type	String
Default	Null

JNDI Name:

The application client run-time uses this field to retrieve configuration information. The name must match the value of the **Name** field on the General tab in the Application Client Resource Reference section of the Application Assembly Tool.

User:

The user ID used, with the **Password** property, for authentication if the calling application does not provide a userid and password explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

The connection factory **User ID** and **Password** properties are used if the calling application does not provide a userid and password explicitly; for example, if the calling application uses the method createTopicConnection(). The JMS client flows the userid and password to the JMS server.

Data type String

Password:

The password used, with the **User ID** property, for authentication if the calling application does not provide a userid and password explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

Data type String
Default Null

Re-Enter Password:

Confirms the password.

Node:

The WebSphere node name of the administrative node where the JMS server runs for this connection factory. Connections created by this factory connect to that JMS server.

Data type Enum
Default Null
Range Pull-down list of nodes in the WebSphere administrative domain.

Application Server:

Enter the name of the application server. This name is not the host name of the machine, but the name of the configured application server.

Port:

Which of the two ports that connections use to connect to the JMS Server. The QUEUED port is for full-function JMS publish/subscribe support, the DIRECT port is for non-persistent, non-transactional, non-durable subscriptions only.

Note: Message-driven beans cannot use the direct listener port for publish/subscribe support. Therefore, any topic connection factory configured with **Port** set to Direct cannot be used with message-driven beans.

Data type Enum
Units Not applicable
Default QUEUED

Range**QUEUED**

The listener port used for full-function JMS-compliant, publish/subscribe support.

DIRECT

The listener port used for direct TCP/IP connection (non-transactional, non-persistent, and non-durable subscriptions only) for publish/subscribe support.

The TCP/IP port numbers for these ports are defined on the WebSphere Internal JMS Server.

Client Id:

The JMS client identifier used for connections to the MQSeries queue manager.

Data type

String

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at runtime for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

WebSphere queue destination settings for application clients

Use this panel to view or change the configuration properties of the selected queue destination for use with the WebSphere JMS provider.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File > Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **JMS Providers > JMS provider instance**. Right-click **WAS Queue Destinations** and click **New**. The following fields appear on the **General** tab.

A queue destination is used to configure the properties of a JMS queue. Connections to the queue are created by the associated queue connection factory for the internal WebSphere JMS provider. A queue for use with the internal WebSphere JMS provider has the following properties.

Name:

The name by which the queue is known for administrative purposes within IBM WebSphere Application Server.

Data type

String

Description:

A description of the queue, for administrative purposes

Data type	String
Default	Null

JNDI Name:

The application client run-time uses this field to retrieve configuration information. The name must match the value of the **Name** field on the General tab in the Application Client Resource Reference section of the Application Assembly Tool.

Persistence:

Whether all messages sent to the destination are persistent, non-persistent, or have their persistence defined by the application

Data type	Enum
Units	Not applicable
Default	APPLICATION_DEFINED
Range	Application defined Messages on the destination have their persistence defined by the application that put them onto the queue. Queue defined [WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties. Persistent Messages on the destination are persistent. Non persistent Messages on the destination are not persistent.

Priority:

Whether the message priority for this destination is defined by the application or the **Specified priority** property

Data type	Enum
Units	Not applicable
Default	APPLICATION_DEFINED

Range**Application defined**

The priority of messages on this destination is defined by the application that put them onto the destination.

Queue defined

[WebSphere MQ destination only]
Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.

Specified

The priority of messages on this destination is defined by the **Specified priority** property. *If you select this option, you must define a priority on the **Specified priority** property.*

Specified Priority:

If the **Priority** property is set to Specified, type here the message priority for this queue, in the range 0 (lowest) through 9 (highest)

If the **Priority** property is set to Specified, messages sent to this queue have the priority value specified by this property.

Data type	Integer
Units	Message priority level
Default	Null
Range	0 (lowest priority) through 9 (highest priority)

Expiry:

Whether the expiry timeout for this queue is defined by the application or the **Specified expiry** property, or messages on the queue never expire (have an unlimited expiry timeout)

Data type	Enum
Units	Not applicable
Default	APPLICATION_DEFINED

Range

Application defined

The expiry timeout for messages on this queue is defined by the application that put them onto the queue.

Specified

The expiry timeout for messages on this queue is defined by the **Specified expiry** property. *If you select this option, you must define a timeout on the **Specified expiry** property.*

Unlimited

Messages on this queue have no expiry timeout, so those messages never expire.

Specified Expiry:

If the **Expiry timeout** property is set to Specified, type here the number of milliseconds (greater than 0) after which messages on this queue expire

Data type

Integer

Units

Milliseconds

Default

Null

Range

Greater than or equal to 0

- 0 indicates that messages never timeout
- Other values are an integer number of milliseconds

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at runtime for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

WebSphere topic destination settings for application clients

Use this panel to view or change the configuration properties of the selected topic destination for use with the internal WebSphere JMS provider.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **JMS Providers** > *JMS provider instance*. Right-click **WAS Topic Destinations** and click **New**. The following fields appear on the **General** tab.

A topic destination is used to configure the properties of a JMS topic for the associated JMS provider. Connections to the topic are created by the associated topic connection factory. A topic for use with the internal WebSphere JMS provider has the following properties.

Name:

The name by which the topic is known for administrative purposes.

Data type String

Description:

A description of the topic, for administrative purposes within IBM WebSphere Application Server.

Data type String

Default Null

JNDI Name:

The application client run-time uses this field to retrieve configuration information. The name must match the value of the **Name** field on the General tab in the Application Client Resource Reference section of the Application Assembly Tool.

Topic Name: The name of the topic as defined to the JMS provider.

Data type String

Persistence:

Whether all messages sent to the destination are persistent, non-persistent, or have their persistence defined by the application

Data type Enum

Units Not applicable

Default APPLICATION_DEFINED

Range

Application defined

Messages on the destination have their persistence defined by the application that put them onto the queue.

Queue defined

[WebSphere MQ destination only]
Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.

Persistent

Messages on the destination are persistent.

Non persistent

Messages on the destination are not persistent.

Priority:

Whether the message priority for this destination is defined by the application or the **Specified priority** property

Data type Enum

Units Not applicable

Default
Range

APPLICATION_DEFINED

Application defined

The priority of messages on this destination is defined by the application that put them onto the destination.

Queue defined

[WebSphere MQ destination only]
Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.

Specified

The priority of messages on this destination is defined by the **Specified priority** property. *If you select this option, you must define a priority on the **Specified priority** property.*

Specified Priority:

If the **Priority** property is set to Specified, type here the message priority for this queue, in the range 0 (lowest) through 9 (highest)

If the **Priority** property is set to Specified, messages sent to this queue have the priority value specified by this property.

Data type	Integer
Units	Message priority level
Default	Null
Range	0 (lowest priority) through 9 (highest priority)

Expiry:

Whether the expiry timeout for this queue is defined by the application or the **Specified expiry** property, or messages on the queue never expire (have an unlimited expiry timeout)

Data type	Enum
Units	Not applicable
Default	APPLICATION_DEFINED

Range

Application defined

The expiry timeout for messages on this queue is defined by the application that put them onto the queue.

Specified

The expiry timeout for messages on this queue is defined by the **Specified expiry** property. *If you select this option, you must define a timeout on the **Specified expiry** property.*

Unlimited

Messages on this queue have no expiry timeout, so those messages never expire.

Specified Expiry:

If the **Expiry timeout** property is set to Specified, type here the number of milliseconds (greater than 0) after which messages on this queue expire

Data type

Integer

Units

Milliseconds

Default

Null

Range

Greater than or equal to 0

- 0 indicates that messages never timeout
- Other values are an integer number of milliseconds

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at runtime for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

MQSeries queue connection factory settings for application clients

Use this panel to view or change the configuration properties of the selected queue connection factory for use with the MQSeries JMS provider. These configuration properties control how connections are created to the associated JMS queue destination.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File > Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **JMS Providers > JMS provider instance**. Right-click **MQ Queue Connection Factories** and click **New**. The following fields appear on the **General** tab.

A queue connection factory creates JMS connections to queue destinations. The queue connection factory is created by the MQSeries JMS provider. A queue connection factory for the MQSeries JMS provider has the following properties.

Note:

- The property values that you specify must match the values that you specified when configuring MQSeries for JMS resources. For more information about configuring MQSeries JMS resources, see the *MQSeries Using Java* book, located in the WebSphere MQ Family library.
- In MQSeries, names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.

Name:

The name by which this queue connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the JMS connection factories across the WebSphere administrative domain.

Data type String

Description:

A description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

JNDI Name:

The application client run-time uses this field to retrieve configuration information. The name must match the value of the **Name** field on the General tab in the Application Client Resource Reference section of the Application Assembly Tool.

User:

The user ID used, with the **Password** property, for authentication if the calling application does not provide a userid and password explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

The connection factory **User ID** and **Password** properties are used if the calling application does not provide a userid and password explicitly; for example, if the calling application uses the method `createQueueConnection()`. The JMS client flows the userid and password to the JMS server.

Data type String

Password:

The password used, with the **User ID** property, for authentication if the calling application does not provide a userid and password explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

Data type String
Default Null

Re-Enter Password:

Confirms the password.

Queue Manager:

The name of the MQSeries queue manager for this connection factory.

Connections created by this factory connect to that queue manager.

Data type String

Host:

The name of the host on which the WebSphere MQ queue manager runs, for client connection only.

Data type String

Default Null

Range A valid TCP/IP hostname

Port:

The TCP/IP port number used for connection to the WebSphere MQ queue manager, for client connection only.

This port must be configured on the WebSphere MQ queue manager.

Data type Integer

Default Null

Range A valid TCP/IP port number, configured on the WebSphere MQ queue manager.

Channel:

The name of the channel used for connection to the WebSphere MQ queue manager, for client connection only.

Data type String

Default Null

Range 1 through 20 ASCII characters

Transport type:

Specifies whether the WebSphere MQ client connection or JNI bindings are used for connection to the WebSphere MQ queue manager. The external JMS provider controls the communication protocols between JMS clients and JMS servers. Tune the transport type when you are using non-ASF nonpersistent, nondurable, nontransactional messaging or when you want to satisfy security issues and the client is local to the queue manager node.

Data type Enum

Units
Default
Range

Not applicable

BINDINGS

BINDINGS

JNI bindings are used to connect to the queue manager. BINDINGS is a shared memory protocol and can only be used when the queue manager is on the same node as the JMS client and comes at some security risks that should be addressed through the use of EJB roles.

CLIENT

WebSphere MQ client connection is used to connect to the queue manager. CLIENT is a typical TCP-based protocol.

DIRECT

For WebSphere MQ Event Broker using DIRECT mode. DIRECT is a lightweight sockets protocol used in nontransactional, nondurable and nonpersistent Publish/Subscribe messaging. DIRECT is only works for clients and message-driven beans using the non-ASF protocol.

QUEUED

QUEUED is a standard TCP protocol.

Recommended

Queue connection factory transport type

BINDINGS is faster by 30% or more, but it lacks security. When you have security concerns, BINDINGS is more desirable than CLIENT.

Topic connection factory transport type

DIRECT is the fastest and should be used where possible. Use BINDINGS when you want to satisfy additional security tasks and the queue manager is local to the JMS client. QUEUED is fallback for all other cases. Note, WebSphere MQ 5.3 before CSD2 with the DIRECT setting can lose messages when used with message-driven beans and under load. This also happens with client-side based applications unless the broker's maxClientQueueSize is set to 0. You can set this to 0 with the command `#wempschangeproperties WAS_nodeName_server1 -e default -o DynamicSubscriptionEngine -n maxClientQueueSize -v 0 -x executionGroupUUID`, where executionGroupUUID can be found by starting the broker and looking in the Event Log/Applications for event 2201. This value is usually ffffffff-0000-0000-000000000000.

Client ID:

The JMS client identifier used for connections to the MQSeries queue manager.

Data type String

CCSID:

The coded character set identifier for use with the WebSphere MQ queue manager.

This coded character set identifier (CCSID) must be one of the CCSIDs supported by WebSphere MQ.

Data type String

For more information about supported CCSIDs, and about converting between message data from one coded character set to another, see the *WebSphere MQ System Administration* and the *WebSphere MQ Application Programming Reference* books. These are available from the WebSphere MQ messaging multiplatform and platform-specific books Web pages; for example, at <http://www-3.ibm.com/software/ts/mqseries/library/manualsa/manuals/platspecific.html>, the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.

Message Retention:

Select this check box to specify that unwanted messages are to be left on the queue. Otherwise, unwanted messages are dealt with according to their disposition options.

Data type	Enum
Units	Not applicable
Default	Cleared
Range	<p>Selected Unwanted messages are left on the queue.</p> <p>Cleared Unwanted messages are dealt with according to their disposition options.</p>

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at runtime for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

MQSeries topic connection factory settings for application clients

Use this panel to view or change the configuration properties of the selected topic connection factory for use with the MQSeries JMS provider. These configuration properties control how connections are created to the associated JMS topic destination.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File > Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **JMS Providers > JMS provider instance**. Right-click **MQ Topic Connection Factories** and click **New**.

A topic connection factory is used to create JMS connections to topic destinations. The topic connection factory is created by the MQSeries JMS provider. A topic connection factory for the MQSeries JMS provider has the following properties.

Note:

- The property values that you specify must match the values that you specified when configuring MQSeries JMS resources. For more information about configuring MQSeries JMS resources, see the *MQSeries Using Java* book.
- In MQSeries, names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.

Name:

The name by which this topic connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the JMS provider.

Data type	String
------------------	--------

Description:

A description of this topic connection factory for administrative purposes within IBM WebSphere Application Server.

Data type	String
Default	Null

JNDI Name:

The JNDI name that is used to bind the topic connection factory into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form *jms/Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform. Although you can edit deployment descriptor files, do not use the administrative console to modify them.

Data type	String
Units	En_US ASCII characters
Default	Null

Range 1 through 45 ASCII characters

User:

The user ID used, with the **Password** property, for authentication if the calling application does not provide a userid and password explicitly.

If you specify a value for the **User** property, you must also specify a value for the **Password** property.

The connection factory **User** and **Password** properties are used if the calling application does not provide a userid and password explicitly; for example, if the calling application uses the method `createTopicConnection()`. The JMS client flows the userid and password to the JMS server.

Data type String

Password:

The password used, with the **User ID** property, for authentication if the calling application does not provide a userid and password explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

Data type String
Default Null

Re-Enter Password:

Confirms the password.

Queue Manager:

The name of the MQSeries queue manager for this connection factory. Connections created by this factory connect to that queue manager.

Data type String

Host:

The name of the host on which the WebSphere MQ queue manager runs, for client connection only.

Data type String
Default Null
Range A valid TCP/IP hostname

Port:

The TCP/IP port number used for connection to the WebSphere MQ queue manager, for client connection only.

This port must be configured on the WebSphere MQ queue manager.

Data type	Integer
Default	Null
Range	A valid TCP/IP port number, configured on the WebSphere MQ queue manager.

Channel:

The name of the channel used for connection to the WebSphere MQ queue manager, for client connection only.

Data type	String
Default	Null
Range	1 through 20 ASCII characters

Transport Type:

Whether MQSeries client connection or JNDI bindings are used for connection to the MQSeries queue manager.

Data type	Enum
Units	Not applicable
Default	BINDINGS
Range	CLIENT MQSeries client connection is used to connect to the MQSeries queue manager. BINDINGS JNDI bindings are used to connect to the MQSeries queue manager.

Client Id:

The JMS client identifier used for connections to the MQSeries queue manager.

Data type	String
------------------	--------

CCSID:

The coded character set identifier for use with the WebSphere MQ queue manager.

This coded character set identifier (CCSID) must be one of the CCSIDs supported by WebSphere MQ.

Data type	String
------------------	--------

Broker Control Queue:

The name of the broker control queue, to which all command messages (except publications and requests to delete publications) are sent

The name of the broker control queue. Publisher and subscriber applications, and other brokers, send all command messages (except publications and requests to

delete publications) to this queue.

Data type	String
Units	En_US ASCII characters
Default	Null
Range	1 through 48 ASCII characters

Broker Queue Manager:

The name of the MQSeries queue manager that provides the Pub/Sub message broker.

Data type	String
Units	En_US ASCII characters
Default	Null
Range	1 through 48 ASCII characters

Broker Pub Queue:

The name of the broker's input queue that receives all publication messages for the default stream

The name of the broker's input queue (stream queue) that receives all publication messages for the default stream. Applications can also send requests to delete publications on the default stream to this queue.

Data type	String
Units	En_US ASCII characters
Default	Null
Range	1 through 48 ASCII characters

Broker Sub Queue:

The name of the broker queue from which non-durable subscription messages are retrieved

The name of the broker's queue from which non-durable subscription messages are retrieved. The subscriber specifies the name of the queue when it registers a subscription.

Data type	String
Units	En_US ASCII characters
Default	Null
Range	1 through 48 ASCII characters

Broker CCSubQ:

The name of the broker's queue from which non-durable subscription messages are retrieved for a ConnectionConsumer. This property applies only for use of the Web container.

The name of the broker queue from which non-durable subscription messages are retrieved for a ConnectionConsumer. This property applies only for use of the Web container.

Data type	String
Units	En_US ASCII characters
Default	Null
Range	1 through 48 ASCII characters

Broker Version:

Specifies whether the message broker is provided by the MQSeries MA0C SupportPac or newer versions of WebSphere message broker products.

Data type	Enum
Units	Not applicable
Default	Advanced
Range	<p>Advanced The message broker is provided by newer versions of WebSphere message broker products (MQ Integrator and MQ Publish and Subscribe)</p> <p>Basic The message broker is provided by the MQSeries MA0C SupportPac (MQSeries - Publish/Subscribe)</p>

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at runtime for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

MQSeries queue destination settings for application clients

Use this panel to view or change the configuration properties of the selected queue destination for use with the MQSeries JMS provider.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File > Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **JMS Providers > JMS provider instance**. Right-click **MQ Queue Destinations** and click **New**. The following fields appear on the **General** tab.

A queue destination configures the properties of a JMS queue. Connections to the queue are created by the associated queue connection factory for the MQSeries JMS provider. A queue for use with the MQSeries JMS provider has the following properties.

Note:

- The property values that you specify must match the values that you specified when configuring MQSeries JMS resources. For more information about configuring MQSeries JMS resources, see the MQSeries *Using Java* book.
- In MQSeries, names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.

Name:

The name by which the queue is known for administrative purposes within IBM WebSphere Application Server.

Data type String

Description:

A description of the queue, for administrative purposes

Data type String

Default Null

JNDI Name:

The application client run-time uses this field to retrieve configuration information. The name must match the value of the **Name** field on the General tab in the Application Client Resource Reference section of the Application Assembly Tool.

Persistence:

Whether all messages sent to the destination are persistent, non-persistent, or have their persistence defined by the application

Data type Enum

Units Not applicable

Default APPLICATION_DEFINED

Range

Application defined

Messages on the destination have their persistence defined by the application that put them onto the queue.

Queue defined

[WebSphere MQ destination only]
Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.

Persistent

Messages on the destination are persistent.

Non persistent

Messages on the destination are not persistent.

Priority:

Whether the message priority for this destination is defined by the application or the **Specified priority** property

Data type Enum

Units Not applicable

Default APPLICATION_DEFINED

Range**Application defined**

The priority of messages on this destination is defined by the application that put them onto the destination.

Queue defined

[WebSphere MQ destination only]
Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.

Specified

The priority of messages on this destination is defined by the **Specified priority** property. *If you select this option, you must define a priority on the **Specified priority** property.*

Specified Priority:

If the **Priority** property is set to Specified, type here the message priority for this queue, in the range 0 (lowest) through 9 (highest)

If the **Priority** property is set to Specified, messages sent to this queue have the priority value specified by this property.

Data type	Integer
Units	Message priority level
Default	Null
Range	0 (lowest priority) through 9 (highest priority)

Expiry:

Whether the expiry timeout for this queue is defined by the application or the **Specified expiry** property, or messages on the queue never expire (have an unlimited expiry timeout)

Data type	Enum
Units	Not applicable
Default	APPLICATION_DEFINED

Range**Application defined**

The expiry timeout for messages on this queue is defined by the application that put them onto the queue.

Specified

The expiry timeout for messages on this queue is defined by the **Specified expiry** property. *If you select this option, you must define a timeout on the **Specified expiry** property.*

Unlimited

Messages on this queue have no expiry timeout, so those messages never expire.

Specified Expiry:

If the **Expiry timeout** property is set to Specified, type here the number of milliseconds (greater than 0) after which messages on this queue expire

Data type

Integer

Units

Milliseconds

Default

Null

Range

Greater than or equal to 0

- 0 indicates that messages never timeout
- Other values are an integer number of milliseconds

Base Queue Name:

The name of the queue to which messages are sent, on the queue manager specified by the **Base queue manager name** property

Data type

String

Base Queue Manager Name:

The name of the MQSeries queue manager to which messages are sent

This queue manager provides the queue specified by the **Base queue name** property.

Data type

String

Units

En_US ASCII characters

Default

Null

Range

A valid MQSeries Queue Manager name, as 1 through 48 ASCII characters

CCSID:

The coded character set identifier for use with the WebSphere MQ queue manager.

This coded character set identifier (CCSID) must be one of the CCSIDs supported by WebSphere MQ.

Data type String

Integer encoding:

If native encoding is not enabled, select whether integer encoding is normal or reversed.

Data type Enum
Units Not applicable
Default NORMAL
Range **NORMAL**
Normal integer encoding is used.
REVERSED
Reversed integer encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Decimal encoding:

If native encoding is not enabled, select whether decimal encoding is normal or reversed.

Data type Enum
Units Not applicable
Default NORMAL
Range **NORMAL**
Normal decimal encoding is used.
REVERSED
Reversed decimal encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Floating point encoding:

If native encoding is not enabled, select the type of floating point encoding.

Data type Enum
Units Not applicable
Default IEEEENORMAL
Range **IEEEENORMAL**
IEEE normal floating point encoding is used.
IEEEEVERSED
IEEE reversed floating point encoding is used.
S390 S390 floating point encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Native encoding:

Select this checkbox to indicate that the queue destination should use native encoding (appropriate encoding values for the Java platform).

Data type	Enum
Units	Not applicable
Default	Cleared
Range	Cleared Native encoding is not used, so specify the properties below for integer, decimal, and floating point encoding. Selected Native encoding is used (to provide appropriate encoding values for the Java platform).

For more information about encoding properties, see the *MQSeries Using Java* document.

Target client:

Whether the receiving application is JMS-compliant or is a traditional WebSphere MQ application

Data type	Enum
Units	Not applicable
Default	MQSeries
Range	MQSeries The target is a non-JMS, traditional WebSphere MQ application. JMS The target is a JMS-compliant application.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at runtime for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

MQSeries topic destination settings for application clients

Use this panel to view or change the configuration properties of the selected topic destination for use with the MQSeries JMS provider.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File > Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **JMS Providers > JMS provider instance**. Right-click **MQ Topic Destinations** and click **New**. The following fields appear on the **General** tab.

A topic destination is used to configure the properties of a JMS topic for the associated JMS provider. Connections to the topic are created by the associated topic connection factory. A topic for use with the MQSeries JMS provider has the following properties.

Note:

- The property values that you specify must match the values that you specified when configuring MQSeries JMS resources. For more information about configuring MQSeries JMS resources, see the *MQSeries Using Java* book.
- In MQSeries, names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.

Name:

The name by which the topic is known for administrative purposes.

Data type String

Description:

A description of the topic, for administrative purposes within IBM WebSphere Application Server.

JNDI Name:

The application client run time uses this field to retrieve configuration information. The name must match the value of the **Name** field on the General tab in the Application Client Resource Reference section of the Application Assembly Tool.

Persistence:

Specifies whether all messages sent to the destination are persistent, non-persistent, or have their persistence defined by the application.

Data type	Enum
Units	Not applicable
Default	APPLICATION_DEFINED
Range	Application defined Messages on the destination have their persistence defined by the application that put them onto the queue. Queue defined [WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties. Persistent Messages on the destination are persistent. Non persistent Messages on the destination are not persistent.

Priority:

Specifies whether the message priority for this destination is defined by the application or the **Specified priority** property.

Data type	Enum
Units	Not applicable
Default	APPLICATION_DEFINED
Range	Application defined The priority of messages on this destination is defined by the application that put them onto the destination. Queue defined [WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties. Specified The priority of messages on this destination is defined by the Specified priority property. <i>If you select this option, you must define a priority on the Specified priority property.</i>

Specified Priority:

If the **Priority** property is set to **Specified**, type the message priority for this queue, in the range 0 (lowest) through 9 (highest).

If the **Priority** property is set to **Specified**, messages sent to this queue have the priority value specified by this property.

Data type	Integer
Units	Message priority level
Default	Null
Range	0 (lowest priority) through 9 (highest priority)

Expiry:

Specifies whether the expiry timeout for this queue is defined by the application or the **Specified expiry** property, or messages on the queue never expire (have an unlimited expiry timeout).

Data type	Enum
Units	Not applicable
Default	APPLICATION_DEFINED

Range**Application defined**

The expiry timeout for messages on this queue is defined by the application that put them onto the queue.

Specified

The expiry timeout for messages on this queue is defined by the **Specified expiry** property. *If you select this option, you must define a timeout on the **Specified expiry** property.*

Unlimited

Messages on this queue have no expiry timeout, so those messages never expire.

Specified Expiry:

If the **Expiry timeout** property is set to **Specified**, type the number of milliseconds (greater than 0) after which messages on this queue expire.

Data type

Integer

Units

Milliseconds

Default

Null

Range

Greater than or equal to 0

- 0 indicates that messages never timeout
- Other values are an integer number of milliseconds

Base Topic Name:

The name of the topic to which messages are sent

Data type

String

CCSID:

The coded character set identifier for use with the WebSphere MQ queue manager.

This coded character set identifier (CCSID) must be one of the CCSIDs supported by WebSphere MQ.

Data type

String

Integer encoding:

If native encoding is not enabled, select whether integer encoding is normal or reversed.

Data type

Enum

Units

Not applicable

Default

NORMAL

Range**NORMAL**

Normal integer encoding is used.

REVERSED

Reversed integer encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Decimal encoding:

If native encoding is not enabled, select whether decimal encoding is normal or reversed.

Data type

Enum

Units

Not applicable

Default

NORMAL

Range**NORMAL**

Normal decimal encoding is used.

REVERSED

Reversed decimal encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Floating point encoding:

If native encoding is not enabled, select the type of floating point encoding.

Data type

Enum

Units

Not applicable

Default

IEEEENORMAL

Range**IEEEENORMAL**

IEEE normal floating point encoding is used.

IEEEEVERSED

IEEE reversed floating point encoding is used.

S390 S390 floating point encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Native encoding:

Select this check box to indicate that the queue destination should use native encoding (appropriate encoding values for the Java platform).

Data type

Enum

Units

Not applicable

Default

Cleared

Range**Cleared**

Native encoding is not used, so specify the properties above for integer, decimal, and floating point encoding.

Selected

Native encoding is used (to provide appropriate encoding values for the Java platform).

For more information about encoding properties, see the *MQSeries Using Java* document.

BrokerDurSubQueue:

The name of the broker queue from which durable subscription messages are retrieved.

The name of the broker queue from which durable subscription messages are retrieved. The subscriber specifies the name of the queue when it registers a subscription.

Data type	String
Units	En_US ASCII characters
Default	Null
Range	1 through 48 ASCII characters

BrokerCCDurSubQueue:

The name of the broker queue from which durable subscription messages are retrieved for a ConnectionConsumer. This property applies only for use of the Web container.

The name of the broker queue from which durable subscription messages are retrieved for a ConnectionConsumer. This property applies only for use of the Web container.

Data type	String
Units	En_US ASCII characters
Default	Null
Range	1 through 48 ASCII characters

Target Client:

Specifies whether the receiving application is JMS-compliant or is a traditional MQSeries application.

Data type	Enum
Units	Not applicable
Default	MQSeries

Range

MQSeries

The target is a non-JMS, traditional MQSeries application.

JMS

The target is a JMS-compliant application.

Custom Properties:

Specifies the name-value pairs for setting additional properties on the object that is created at runtime for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Generic JMS connection factory settings for application clients

Use this panel to view or change the configuration properties of the selected JMS connection factory for use with the associated JMS provider. These configuration properties control how connections are created to the associated JMS destination. To view this Application Client Resource Configuration Tool (ACRCT) page, click **File > Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **JMS Providers > new JMS Provider instance**. Right click **JMS Connection Factories > click New**. The following fields appear on the **General** tab.

A JMS connection factory creates connections to JMS destinations. The JMS connection factory is created by the associated JMS provider. A JMS connection factory for a generic JMS provider (other than the internal WebSphere JMS provider or the MQSeries JMS provider) has the following properties:

Name:

The name by which this JMS connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the associated JMS provider.

Data type

String

Description:

A description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type

String

Default

Null

JNDI Name:

The application client run-time uses this field to retrieve configuration information. The name must match the value of the **Name** field on the General tab in the Application Client Resource Reference section of the Application Assembly Tool.

User:

The user ID used, with the **Password** property, for authentication if the calling application does not provide a userid and password explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

The connection factory **User ID** and **Password** properties are used if the calling application does not provide a userid and password explicitly; for example, if the calling application uses the method `createQueueConnection()`. The JMS client flows the userid and password to the JMS server.

Data type String

Password:

The password used, with the **User ID** property, for authentication if the calling application does not provide a userid and password explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

Data type String

Default Null

Re-Enter Password:

Confirms the password entered in the Password field.

External JNDI Name:

The JNDI name that is used to bind the queue into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form `jms/Name`, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform. Although you can edit deployment descriptor files, do not use the administrative console to modify them.

Data type String

Connection Type:

Whether this JMS destination is a queue (for point-to-point) or topic (for pub/sub).

Select one of the following options:

Queue

A JMS queue destination for point-to-point messaging.

Topic A JMS topic destination for pub/sub messaging.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at runtime for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Generic JMS destination settings for application clients

Use this panel to view or change the configuration properties of the selected JMS destination for use with the associated JMS provider.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File > Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **JMS Providers** > *new JMS Provider instance*. Right click **JMS Destinations** > click **New**. The following fields appear on the **General** tab.

A JMS destination is used to configure the properties of a JMS destination for the associated generic JMS provider. Connections to the JMS destination are created by the associated JMS connection factory. A JMS destination for use with a generic JMS provider (not the internal WebSphere JMS provider or MQSeries JMS provider) has the following properties.

Name:

The name by which the queue is known for administrative purposes within IBM WebSphere Application Server.

Data type String

Description:

A description of the queue, for administrative purposes

JNDI Name:

The JNDI name of the actual (physical) name of the JMS destination bound into JNDI.

External JNDI Name:

The JNDI name that is used to bind the queue into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form *jms/Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Destination Type:

Whether this JMS destination is a queue (for point-to-point) or topic (for pub/sub).

Select one of the following options:

Queue

A JMS queue destination for point-to-point messaging.

Topic A JMS topic destination for pub/sub messaging.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at runtime for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Example: Configuring JMS Provider, JMS Connection Factory and JMS Destination settings for application clients

The purpose of this article is to help you to configure JMS Provider, JMS Connection Factory and JMS Destination settings.

- Required fields:
 - JMS Provider Properties page: name, and at least one protocol provider
 - JMS Connection Factory Properties page: name, jndiName, destination type
 - JMS Destination Properties page: name, jndiName, destination type
- Special cases:
 - The destination type must be QUEUE, or TOPIC.
- Example (shown here on multiple lines for publication):

```
<resources.jms:JMSProvider xmi:id="JMSProvider_3" name="genericJMSProvider:name"
  description="genericJMSProvider:description"
  externalInitialContextFactory="genericJMSProvider:contextFactoryClass"
  externalProviderURL="genericJMSProvider:providerUrl">
<classpath>genericJMSProvider:classpath</classpath>
<factories xmi:type="resources.jms:GenericJMSDestination"
  xmi:id="GenericJMSDestination_1" name="jmsDestination:name"
  jndiName="jmsDestination:jndiName" description="jmsDestination:description"
  externalJNDIName="jmsDestination:externalJndiName" type="QUEUE">
<propertySet xmi:id="J2EEResourcePropertySet_15">
<resourceProperties xmi:id="J2EEResourceProperty_17"
  name="jmsDestination:customName" value="jmsDestination:customValue"/>
</propertySet>
</factories>
<factories xmi:type="resources.jms:GenericJMSConnectionFactory"
  xmi:id="GenericJMSConnectionFactory_1" name="jmsCF:name"
  jndiName="jmsCF:jndiName" description="jmsCF:description" userID="jmsCF:user"
  password="{xor}NTIsHB11MT4y0g==" externalJNDIName="jmsCF:externalJndiName"
  type="QUEUE">
<propertySet xmi:id="J2EEResourcePropertySet_16">
<resourceProperties xmi:id="J2EEResourceProperty_18" name="jmsCF:customName"
  value="jmsCF:customValue"/>
</propertySet>
</factories>
<propertySet xmi:id="J2EEResourcePropertySet_17">
<resourceProperties xmi:id="J2EEResourceProperty_19"
  name="genericJMSProvider:customName" value="genericJMSProvider:customValue"/>
</propertySet>
</resources.jms:JMSProvider>
```

Configuring new connection factories for application clients

During this task, you create a new JMS connection factory configuration for your application client.

1. Click the JMS provider for which you want to create a connection factory in the tree. Do one of the following:
 - Configure a new JMS provider.
 - Click an existing JMS provider.
2. Expand the JMS provider to view its **JMS Connection Factories** folder.
3. Click the folder. Do one of the following:
 - Right-click the folder and click **New Factory**.
 - Click **Edit > New** on the menu bar.
4. Configure the JMS connection factory properties in the resulting property dialog.
5. Click **OK** when you finish.
6. Click **File > Save** on the menu bar to save your changes.

Configuring new Java Message Service destinations for application clients

During this task, you create new Java Message Service (JMS) destination configuration for your application client.

1. Click the JMS provider in the tree for which you want to create a destination. Do one of the following:
 - Configure a new JMS provider.
 - Click an existing JMS provider.
2. Expand the JMS provider to view its **JMS Destinations** folder.
3. Click the folder. Do one of the following:
 - Right-click the folder and click **New Factory**.
 - Click **Edit > New** on the menu bar.
4. Configure the JMS destination properties in the resulting property dialog.
5. Click **OK** when you finish.
6. Click **File > Save** on the menu bar to save your changes.

Example: Configuring MQ Queue and Topic connection factories and destination factories for application clients

The purpose of this article is to help you configure MQ Queue connection factory, MQ Topic connection factory, MQ Queue destination factory, and MQ Topic destination factory settings.

- Required fields:
 - MQ Queue Connection Factory Properties page: name, jndiName, transport type
 - MQ Topic Connection Factory Properties page: name, jndiName, broker version
 - MQ Queue Factory Properties page: name, jndiName, persistence, priority, expiry, baseQueueName, targetClient
 - MQ Topic Factory Properties page: name, jndiName, persistence, priority, expiry, baseQueueName, targetClient
- Special cases:
 - The transport type must be CLIENT, or BINDINGS.
 - The Broker Version must be MA0C, or MQSI.
 - The port must be a numerical value between -2417483648 and 2417483647.
 - The CCSID must be a numerical value between -2417483648 and 2417483647.
 - The persistence value must be APPLICATION_DEFINED, QUEUE_DEFINED, PERSISTENT or, NONPERSISTENT.
 - The priority must be APPLICATION_DEFINED, QUEUE_DEFINED, or SPECIFIED.

- The expiry must be APPLICATION_DEFINED, UNLIMITED, or SPECIFIED.
- The integer encoding must be Normal, or Reversed.
- The decimal encoding must be Normal, or Reversed.
- The floating encoding must be IEEENormal, IEEEReversed, S390.
- The target client must be JMS or MQ.
- On the MQ Queue Connection Factory Properties page, only set the queueManager, host, and portWhen (required) fields if the transport type is CLIENT.
- On the MQ Topic Connection Factory Properties page, only set the queueManager, host, and port (required) fields if the transport type is CLIENT.
- On the the MQ Topic Factory Properties, and the MQ Queue Factory Properties pages, only set the Integer encoding, decimal encoding, and floating point encoding (required) fields if you do not set nativeEncoding.
- On the MQ Topic Factory Properties, and the MQ Queue Factory Properties pages, the specified priority entry field must be an integer between 0 and 9 if priority is set to SPECIFIED .
- On the the MQ Topic Factory Properties, and the MQ Queue Factory Properties pages, the specified expiry entry field must be a value greater than 0 if expiry is set to SPECIFIED.

- Example (shown here on multiple lines for publication):

```

<resources.jms:JMSProvider xmi:id="JMSProvider_1" name="MQ JMS Provider"
  description="mqJMSProvider:description"
  externalInitialContextFactory="mqJMSProvider:contextFactoryClass"
  externalProviderURL="mqJMSProvider:providerUrl">
<classpath>mqJMSProvider:classpath</classpath>
<factories xmi:type="resources.jms.mqseries:MQQueueConnectionFactory"
  xmi:id="MQQueueConnectionFactory_1" name="mqQCF:name"
  jndiName="mqQCF:jndiName" description="mqQCF:description"
  userID="mqQCF:user" password="{xor}Mi40HB11MT4y0g=="
  queueManager="mqQCF:queueManager" host="mqQCF:host" port="1"
  channel="mqQCF:channel" transportType="CLIENT" clientID="mqQCF:clientId"
  CCSID="2">
<propertySet xmi:id="J2EEResourcePropertySet_3">
<resourceProperties xmi:id="J2EEResourceProperty_3" name="mqQCF:customName"
  value="mqQCF:customValue"/>
</propertySet>
</factories>
<factories xmi:type="resources.jms.mqseries:MQTopicConnectionFactory"
  xmi:id="MQTopicConnectionFactory_1" name="mqTCF:name"
  jndiName="mqTCF:jndiName" description="mqTCF:description"
  userID="mqTCF:user" password="{xor}Mi4LHB11NTE7NhE+Mjo="
  host="mqTCF:host" port="1" transportType="CLIENT" channel="mqTCF:channel"
  queueManager="mqTCF:queueManager"
  brokerControlQueue="mqTCF:brokerControlQueue"
  brokerQueueManager="mqTCF:brokerQueueManager"
  brokerPubQueue="mqTCF:brokerPubQueue"
  brokerSubQueue="mqTCF:brokerSubQueue" brokerCCSubQ="mqTCF:brokerCCSubQ"
  brokerVersion="MA0C" clientID="mqTCF:clientId" CCSID="2">
<propertySet xmi:id="J2EEResourcePropertySet_4">
<resourceProperties xmi:id="J2EEResourceProperty_4" name="mqTCF:customName"
  value="mqTCF:customValue"/>
</propertySet>
</factories>
<factories xmi:type="resources.jms.mqseries:MQQueue" xmi:id="MQQueue_1"
  name="mqQ:name" jndiName="mqQ:jndiName" description="mqQ:description"
  persistence="APPLICATION_DEFINED" priority="SPECIFIED"
  specifiedPriority="1" expiry="SPECIFIED" specifiedExpiry="1"
  baseQueueName="mqQ:baseQueueName"
  baseQueueManagerName="mqQ:baseQueueManagerName" CCSID="1"
  integerEncoding="Normal" decimalEncoding="Normal"
  floatingPointEncoding="IEEENormal" targetClient="JMS">
<propertySet xmi:id="J2EEResourcePropertySet_5">
<resourceProperties xmi:id="J2EEResourceProperty_5" name="mqQ:customName"

```

```

    value="mqQ:customValue"/>
</propertySet>
</factories>
<factories xmi:type="resources.jms.mqseries:MQTopic" xmi:id="MQTopic_1"
  name="mqT:name" jndiName="mqT:jndiName" description="mqT:description"
  persistence="APPLICATION_DEFINED" priority="SPECIFIED"
  specifiedPriority="1" expiry="SPECIFIED" specifiedExpiry="2"
  baseTopicName="mqT:baseTopicName" CCSID="3" integerEncoding="Normal"
  decimalEncoding="Normal" floatingPointEncoding="IEEENormal"
  targetClient="JMS" brokerDurSubQueue="mqT:brokerDurSubQueue"
  brokerCCDurSubQueue="mqT:brokerCCDurSubQueue">
<propertySet xmi:id="J2EEResourcePropertySet_6">
<resourceProperties xmi:id="J2EEResourceProperty_6" name="mqT:customName"
  value="mqT:customValue"/>
</propertySet>
</factories>
<propertySet xmi:id="J2EEResourcePropertySet_7">
<resourceProperties xmi:id="J2EEResourceProperty_7"
  name="mqJMSProvider:customName" value="mqJMSProvider:customValue"/>
</propertySet>
</resources.jms:JMSProvider>

```

Example: Configuring WAS Queue and Topic connection factories and destination factories for application clients

The purpose of this article is to help you to configure WAS Queue connection factory, WAS Topic connection factory, WAS Queue destination factory, and WAS Topic destination factory settings.

- Required fields:
 - JMS Provider Properties page: name
 - WAS Queue Connection Factory Properties page: name, jndiName, node
 - WAS Topic Connection Factory Properties page: name, jndiName, node, port
 - WAS Queue Factory Properties page: name, jndiName, node, persistence, priority, expiry
 - WAS Topic Factory Properties page: name, jndiName, topic name, persistence, priority, expiry
- Special cases:
 - The port must be QUEUED or DIRECT.
 - The CCSID must be a numerical value between -2417483648 and 2417483647.
 - The persistence value must be APPLICATION_DEFINED, PERSISTENT, or NONPERSISTENT.
 - The priority must be APPLICATION_DEFINED, or SPECIFIED.
 - The expiry must be APPLICATION_DEFINED, UNLIMITED, or SPECIFIED.
 - On the WAS Topic Factory Properties, and the WAS Queue Factory Properties pages, the specified priority entry field must be an integer between 0 and 9 if priority is set to SPECIFIED .
 - On the WAS Topic Factory Properties, and the WAS Queue Factory Properties pages, the specified expiry entry field must be an value greater than 0 if expiry is set to SPECIFIED.
- Example (shown here on multiple lines for publication):

```

<resources.jms:JMSProvider xmi:id="JMSProvider_2" name="WebSphere
  JMS Provider" description="wasJMSProvider:description"
  externalInitialContextFactory="wasJMSProvider:contextfactoryclass"
  externalProviderURL="wasJMSProvider:providerUrl">
<classpath>wasJMSProvider:classpath</classpath>
<factories xmi:type="resources.jms.internalmessaging:WASQueueConnectionFactory"
  xmi:id="WASQueueConnectionFactory_1" name="wasQCF:name"
  jndiName="wasQCF:jndiName" description="wasQCF:description"
  userID="wasQCF:user" password="{xor}KD4sDhwZZS0i0=" node="wasQCF:Node">
<propertySet xmi:id="J2EEResourcePropertySet_8">
<resourceProperties xmi:id="J2EEResourceProperty_8" name="wasQCF:customName"

```

```

        value="wasQCF:customValue"/>
    </propertySet>
</factories>
<factories xmi:type="resources.jms.internalmessaging:WASTopicConnectionFactory"
    xmi:id="WASTopicConnectionFactory_1" name="wasTCF:name"
    jndiName="wasTCF:jndiName" description="wasTCF:description"
    userID="wasTCF:user" password="{xor}KD4sCwxZZTE+Mjo=" node="wasTCF:node"
    port="QUEUED" clientID="wasTCF:clientId">
<propertySet xmi:id="J2EEResourcePropertySet_9">
<resourceProperties xmi:id="J2EEResourceProperty_9" name="wasTCF:customName"
    value="wasTCF:customValue"/>
</propertySet>
</factories>
<factories xmi:type="resources.jms.internalmessaging:WASQueue"
    xmi:id="WASQueue_1" name="wasQ:name" jndiName="wasQ:jndiName"
    description="wasQ:description" node="wasQ:node"
    persistence="APPLICATION_DEFINED" priority="SPECIFIED"
    specifiedPriority="1" expiry="SPECIFIED" specifiedExpiry="1">
<propertySet xmi:id="J2EEResourcePropertySet_10">
<resourceProperties xmi:id="J2EEResourceProperty_10" name="wasQ:customName"
    value="wasQ:customValue"/>
</propertySet>
</factories>
<factories xmi:type="resources.jms.internalmessaging:WASTopic"
    xmi:id="WASTopic_1" name="wasT:name" jndiName="wasT:jndiName"
    description="wasT:description" topic="wasT:topicName"
    persistence="APPLICATION_DEFINED" priority="SPECIFIED"
    specifiedPriority="1" expiry="SPECIFIED" specifiedExpiry="1">
<propertySet xmi:id="J2EEResourcePropertySet_11">
<resourceProperties xmi:id="J2EEResourceProperty_11" name="wasT:customName"
    value="wasT:customValue"/>
</propertySet>
</factories>
<propertySet xmi:id="J2EEResourcePropertySet_12">
<resourceProperties xmi:id="J2EEResourceProperty_12"
    name="wasJMSProvider:customName" value="wasJMSProvider:customValue"/>
</propertySet>
</resources.jms:JMSProvider>

```

Configuring new resource environment providers for application clients

During this task, you create new resource environment provider configurations for your application client.

To configure a new resource environment provider, perform the following steps:

1. Start the tool and open the EAR file for which you want to configure the new JMS provider. The EAR file contents display in a tree view.
2. Select from the tree the JAR file in which you want to configure the new JMS provider.
3. Expand the JAR file to view its contents.
4. Click the folder called **Resource Environment Providers**. Do one of the following:
 - Right-click the folder and click **New Provider**.
 - Click **Edit > New** on the menu bar.
5. Configure the JMS provider properties in the resulting property dialog.
6. Click **OK** when finished.
7. Click **File > Save** on the menu bar to save your changes.

Resource environment provider settings for application clients

Use this page to specify resource environment entry properties.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File > Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file. Right-click **Resource Environment Providers >** and click **New**. The following fields appear on the **General** tab:

Name:

Specifies the administrative name for the resource environment provider.

Description:

Specifies a description of the resource environment provider for your administrative records.

Class Path:

Specifies the path to the JAR file that contains the implementation classes for the resource environment provider.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at runtime for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Configuring new resource environment entries for application clients

During this task, you create new resource environment entries for your client application.

1. Start the Application Client Resource Configuration Tool (ACRCT).
2. Open the EAR file for which you want to configure the new resource environment entry. The EAR file contents are in the displayed tree view.
3. Click the desired resource environment provider, and complete the following action to configure new providers:
 - Configure a new resource environment provider.
4. Expand the resource environment provider to view the **resource environment entries** folder.
5. Click the folder. Complete one of the following actions:
 - Right-click the folder and select **New Factory**.
 - Click **Edit > New** on the menu bar.
6. Configure the data source properties in the resulting property dialog.
7. Click **OK**.
8. Click **File > Save** on the menu bar to save your changes.

Resource environment entry settings for application clients

Use this page to specify resource environment entry properties.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File > Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Resource Environment Providers > resource environment instance**. Right-click **Resource environment entry >** and click **New**. The following fields appear on the **General** tab:

Name:

Specifies the administrative name for the resource environment entry.

Description:

Specifies a description of the URL for your administrative records.

JNDI Name:

Specifies the Java Naming and Directory Interface (JNDI) name for the resource, including any naming subcontexts.

Use this name to link to the binding information of the platform. The binding associates the resources defined in the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform. Although you can edit deployment descriptor files, do not use the administrative console to modify them.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at runtime for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Managing application clients

Perform the following tasks after deploying application clients.

After deploying application clients on z/OS or OS/390, you might want to or need to update the resources that you configured for those clients. To do so, you may complete one of the following tasks:

- Run the Application Client Resource Configuration Tool (ACRCT) on Windows, according to the steps below, and then reinstall the application on z/OS; or
- Run the ACRCT scripting tool on z/OS.

Note: This task only applies to J2EE application clients.

1. Update data source and data source provider configurations.
2. Update URLs and URL provider configurations.
3. Update mail session configurations.
4. Update JMS provider, connection factories, and destination configurations.
5. Update MQ JMS provider, MQ connection factories, and MQ destination configurations.

6. Update Resource Environment Entry and Resource Environment Provider configurations.
7. (Optional) Remove application client resources.

Updating data source and data source provider configurations with the Application Client Resource Configuration Tool

During this task, you update the configuration of an existing data source or data source provider.

1. Start the tool and open the EAR file containing the data source or data source provider. The EAR file contents display in a tree view.
2. Select from the tree the JAR file containing the data source or data source provider to update.
3. Expand the JAR file to view its contents until you locate the particular data source or data source provider to update. Do one of the following:
 - Right-click the object and click **Properties**.
 - Click **Edit > Properties** on the menu bar.
4. Update the properties in the resulting property dialog. For detailed field help, go to:
 - Data source provider properties
 - Data source properties
5. Click **OK** when finished.
6. Click **File > Save** on the menu bar to save your changes.

Updating URLs and URL provider configurations for application clients

1. Start the tool and open the EAR file containing the URL or URL provider. The EAR file contents display in a tree view.
2. Select from the tree the JAR file containing the URL or URL provider to update.
3. Expand the JAR file to view its contents.
4. Keep expanding the JAR file contents until you locate the particular URL or URL provider to update. Do one of the following:
 - a. Right-click the object and click **Properties**
 - b. Click **Edit > Properties** on the menu bar.
5. Update the properties in the resulting property dialog.
6. Click **OK** when finished.
7. Click **File > Save** to save your changes on the menu bar.

Updating mail session configurations for application clients

During this task, you update the configuration of an existing JavaMail session.

Note:

You cannot update the name of the default JavaMail provider. Also, you cannot delete the default JavaMail provider from the tree.

1. Start the tool and open the EAR file containing the JavaMail session. The EAR file contents display in a tree view.
2. Select from the tree the JAR file containing the JavaMail session to update.
3. Expand the JAR file to view its contents.

4. Keep expanding the JAR file contents until you locate the particular JavaMail session to update. Do one of the following:
 - a. Right-click the object and click **Properties**
 - b. Click **Edit > Properties** from the menu bar.
5. Update the properties in the resulting property dialog.
6. Click **OK** when finished.
7. Select **File > Save** from the menu bar to save your changes.

Updating Jave Message Service provider, connection factories, and destination configurations for application clients

During this task, you update the configuration of an existing Java Message Service (JMS) provider, connection factory, or destination.

1. Start the tool and open the EAR file containing the JMS provider, connection factory, or destination. The EAR file contents display in a tree view.
2. Select from the tree the JAR file containing the JMS provider, connection factory, or destination to update.
3. Expand the JAR file to view its contents until you locate the particular JMS provider, connection factory, or destination to update. When you find it, do one of the following:
 - Right-click the object and click **Properties**.
 - Click **Edit > Properties** on the menu bar.
4. Update the properties in the resulting property dialog. For detailed field help, see:
 - JMS provider properties
 - WAS Queue connection factory properties
 - WAS Topic connection factory properties
 - WAS Queue destination properties
 - WAS Topic destination properties
5. Click **OK**.
6. Click **File > Save** to save your changes.

Updating MQ Java Message Service provider, MQ connection factories, and MQ destination configurations for application clients

During this task, you will update the configuration of an existing MQ JMS provider, MQ connection factory, or MQ destination.

1. Start the Application Client Resource Configuration Tool (ACRCT).
2. Open the EAR file containing the MQ JMS provider, MQ connection factory, or MQ destination. The EAR file contents are displayed in a tree view.
3. Select the JAR file containing the MQ JMS provider, MQ connection factory, or MQ destination to update.
4. Expand the JAR file to view its contents until you locate the particular MQ JMS provider, MQ connection factory, or MQ destination that you want to update. Complete one of the following actions:
 - Right-click the object and click **Properties**.
 - Click **Edit > Properties** on the menu bar.

5. Update the properties in the resulting property dialog. For detailed field help, see:
 - JMS provider properties
 - MQ Queue connection factory properties
 - MQ Topic connection factory properties
 - MQ Queue destination properties
 - MQ Topic destination properties
6. Click **OK**.
7. Click **File > Save** to save your changes.

Updating Resource Environment Entry and Resource Environment Provider configurations for application clients

During this task, you update the configuration of an existing Resource Environment Entry or Resource Environment Provider.

1. Start the tool and open the EAR file containing the Resource Environment Entry or Resource Environment Provider. The EAR file contents display in a tree view.
2. Select from the tree the JAR file containing the Resource Environment Entry or Resource Environment provider to update.
3. Expand the JAR file to view its contents until you locate the Resource Environment Entry or Resource Environment Provider to update. Do one of the following:
 - Right-click the object and click **Properties**.
 - Click **Edit > Properties** on the menu bar.
4. Update the properties in the resulting property dialog. For detailed field help, see:
 - Resource environment provider properties
 - Resource environment entry properties
5. Click **OK** when you finish.
6. Click **File > Save** on the menu bar to save your changes.

Example: Configuring Resource Environment settings

The purpose of this article is to help you configure Resource Environment settings.

- Required fields:
 - Resource Environment Provider page: name
 - Resource Environment Entry page: name, jndiName
- Example:

```
<resources.env:ResourceEnvironmentProvider xmi:id="ResourceEnvironmentProvider_1" name="resourceEnvProvider"
<classpath>resourceEnvProvider:classpath</classpath>
<factories xmi:type="resources.env:ResourceEnvEntry" xmi:id="ResourceEnvEntry_1" name="resourceEnvEntry">
<propertySet xmi:id="J2EEResourcePropertySet_20">
<resourceProperties xmi:id="J2EEResourceProperty_22" name="resourceEnvEntry:customName" value="resourceEnvEntry:customName"
</propertySet>
</factories>
<propertySet xmi:id="J2EEResourcePropertySet_21">
<resourceProperties xmi:id="J2EEResourceProperty_23" name="resourceEnvProvider:customName" value="resourceEnvProvider:customName"
</propertySet>
</resources.env:ResourceEnvironmentProvider>
```

Example: Configuring Resource Environment custom settings for application clients

The purpose of this article is to help you configure Resource Environment custom settings.

- The custom page applies to every resource type. You can specify as many custom names and values as you need.
- Example:

```
<propertySet xmi:id="J2EEResourcePropertySet_20">
<resourceProperties xmi:id="J2EEResourceProperty_22" name="resourceEnvEntry:customName" value="res
</propertySet>
```

Removing application client resources

Note: This task only applies to J2EE application clients.

To remove resources for application clients running on z/OS or OS/390, you may complete one of the following tasks:

- Run the Application Client Resource Configuration Tool (ACRCT) on Windows, according to the steps below, and then reinstall the application on z/OS; or
 - Run the ACRCT scripting tool on z/OS.
1. Start the Application Client Resource Configuration Tool (ACRCT) and open the EAR file from which you want to remove an object. The EAR file contents display in a tree view. If you already have an EAR file open, and have made some changes, click **File** > **Save** to save your work before proceeding to delete an object.
 2. Locate the object that you want to remove in the tree.
 3. Right-click the object, then click **Delete**.
 4. Click **File** > **Save**.

The option to delete an item does not offer a confirmation dialog. As a safeguard, consider saving your work right before you begin this task. If you change your mind after removing an item, you can close the EAR file without saving your changes, canceling your deletion. Remember to close the EAR file immediately after the deletion, or you also lose any unsaved work that you performed since the deletion.

Running application clients

The J2EE specification requires support for a client container that runs stand-alone Java applications (known as J2EE application clients) and provides J2EE services to the applications. J2EE services include naming, security, and resource connections.

You are ready to run your application client using this tool after you have:

1. Written the application client program.
2. Assembled and installed an application module (.ear file) in the application server run time.
3. Deployed the application using the Application Client Resource Configuration Tool (ACRCT) on Windows, or the ACRCT scripting tool on z/OS.

Note: This task only applies to J2EE application clients.

1. Open a command window and invoke the following script to launch J2EE application clients using the launchClient shell:

```
install_root/bin/launchClient.bat
```

The launchClient batch command starts the application client run time, which:

- Initializes the client run time.
- Loads the class that you designated as the main class with the Application Assembly Tool (AAT).

- Runs the main method of the application client program.

When your program terminates, the application client run time cleans up the environment and the Java Virtual Machine code ends.

2. On z/OS or OS/390, invoke the following script to launch J2EE application clients using the launchClient shell:

```
install_root/bin/launchClient.bat
```

Example invocation on z/OS:

```
/usr/lpp/WebSphere/V5R0M0/bin/launchClient.bat
```

The launchClient batch command:

- Initializes the client run time.
- Loads the class that you designated as the main class with the Application Assembly Tool (AAT).
- Runs the main method of the application client program.

When your program terminates, the application client run time cleans up the environment and the Java Virtual Machine code ends.

3. Pass parameters to the **launchClient** command. You can pass parameters to your application client program as well. The launchClient command allows you to do both. The launchClient command requires that the first parameter is either:
 - An EAR file specifying the application client to launch.
 - A request for launchClient usage information.

All other parameters intended for the **launchClient** command must begin with the -CC prefix.

Parameters that are not EAR files, or usage requests, or that do not begin with the -CC prefix, are ignored by the application client run time, and are passed directly to the application client program.

The **launchClient** command retrieves parameters from three places:

- a. The command line
- b. A properties file
- c. System properties

The parameters are resolved in the order listed above, with command line values having the highest priority and system properties the lowest. This prioritization allows you to set and override default values.

4. Specify the server name. By default, the **launchClient** command uses the environment variable `COMPUTERNAME` for the `BootstrapHost` property value. This setting is effective for testing your application client when it is installed on the same computer as the server. However, in other cases override this value with the name of your server.

You can override the `BootstrapHost` value by invoking `launchClient` with the following parameters:

```
launchClient myapp.ear -CCBootstrapHost=abc.midwest.mycompany.com
```

You can also override the default by specifying the value in a properties file and passing the file name to the launchClient shell.

Note: Security is controlled by the server. You do not need to configure security on the client because the client assumes that security is enabled. If security is not enabled, the server ignores the security request, and the application client works as expected.

You can store `launchClient` values in a properties file, a good method for distributing default values. You can then override one or more values on the command line. The format of the file is one `launchClient -CC` parameter per line without the `-CC` prefix. For example (shown here on multiple lines for publication):

```
verbose=true classpath=c:\mydir\util.jar;c:\mydir\harness.jar;c:\production\G19\global.jar
BootstrapHost=abc.westcoast.mycompany.com tracefile=c:\WebSphere\mylog.txt

verbose=true
classpath=/usr/lpp/mydir/util.jar;/usr/lpp/mydir/harness.jar;/usr/lpp/production/G19/global.jar
BootstrapHost=abc.westcoast.mycompany.com tracefile=/usr/lpp/WebSphere/mylog.txt
```

launchClient tool

This section describes the command line syntax for the Java TM2 Platform, WebSphere Application Server Enterprise (J2EE) `launchClient` tool. You can use the `launchClient` command from a node within a Network Deployment environment. **However, do not attempt to use the `launchClient` command from Deployment Manager.**

The command line invocation syntax for the `launchClient` tool follows:

```
launchClient [<userapp.ear> [-help|-?] [-CCname=value] [app args]
```

where *userapp.ear* is the path and the name of the EAR file that contains the application client, *name* is the name of the parameter, *value* is the value to which the parameter ID is set, and *app args* are arguments that pass to the application client.

To print the usage information, the first parameter must be a path and a name to an EAR file, `-help`, or `-?`. All other parameters are optional and can appear in any order. The application client run time ignores any optional parameters that do not begin with a `-CC` prefix, and passes them to the application client.

Parameters

Supported arguments include:

-CCsoapConnectorPort

The soap connector port. If you do not specify this argument, the WebSphere Application Server default value is used.

-CCverbose

This option displays additional information messages. The default is `false`.

-CCclasspath

A class path value. When you launch an application, the system class path is not used. If you want to access classes that are not in the EAR file or part of the resource class paths, specify the appropriate class path here. Multiple paths can be concatenated.

-CCjar

The name of the client JAR file that resides within the EAR file for the application you wish to launch. Use this argument when you have multiple client JAR files in the EAR file.

-CCadminConnectorHost

Specifies the host name of the server from which configuration information is retrieved. The default is the value of the `-CCBootstrapHost` parameter or the value of the local host if the `-CCBootstrapHost` parameter is not specified.

-CCadminConnectorPort

Indicates the port number that the administrative client function should use. The default value is 8880 for SOAP connections and 2809 for RMI connections.

-CCadminConnectorType

Specifies how the administrative client should connect to the server. Specify RMI to use the RMI connection type or specify SOAP to use the SOAP connection type. The default value is SOAP.

-CCadminConnectorUser

Administrative clients use this user name when a server requires authentication. If the connection type is SOAP, and security is enabled on the server, this parameter is required. The SOAP connector does not prompt for authentication.

-CCadminConnectorPassword

The password for the user name that the `-CCadminConnectorUser` parameter specifies.

-CCaltDD

The name of an alternate deployment descriptor file. This parameter is used with the `-CCjar` parameter to specify the deployment descriptor to use. Use this argument when a client jar file is configured with more than one deployment descriptor. Set the value to null to use the client JAR file standard deployment descriptor. Although you can edit deployment descriptor files, do not use the administrative console to modify them.

-CCBootstrapHost

The name of the host server you want to connect to initially. The format is:
your.server.ofchoice.com

-CCBootstrapPort

The server port number. If you do not specify this argument, the WebSphere Application Server default value is used.

-CCproviderURL

Provides bootstrap server information that the initial context factory can use to obtain an initial context. WebSphere Application Server initial context factory can use either a CORBA object URL or an IIOP URL. CORBA object URLs are more flexible than IIOP URLs and are the recommended URL format to use. This value can contain more than one bootstrap server address. This feature can be used when attempting to obtain an initial context from a server cluster. You can specify bootstrap server addresses, for all servers in the cluster, in the URL. The operation will succeed if at least one of the servers is running, eliminating a single point of failure. The address list does not process in a particular order. For naming operations, this value overrides the `-CCBootstrapHost` and `-CCBootstrapPort` parameters. An example of a CORBA object URL specifying multiple systems follows (shown here on multiple lines for publication):

```
-CCproviderURL=corbaloc:iiop:myserver.mycompany.com:9810,  
:mybackupserver.mycompany.com:2809
```

This value is mapped to the `java.naming.provider.url` system property.

-CCinitonly

Use this option to initialize application client run time for ActiveX application clients without launching the client application. The default is false.

-CCtrace

Use this option to obtain debug trace information. You might need this information when reporting a problem to IBM Service. The default is false.

-CCtracefile

The name of the file to write trace information. The default is to output to the console.

-CCpropfile

Name of a properties file that contains launchClient properties. Specify the properties without the -CC prefix in the file. For example: verbose=true.

-CCsecurityManager

Enables and runs the WebSphere Application Server with a security manager. The default is disable.

-CCsecurityMgrClass

The fully qualified name of a class that implements a security manager. Only use this argument if the -CCsecurityManager parameter is set to enable. The default is java.lang.SecurityManager.

-CCsecurityMgrPolicy

The name of a security manager policy file. Only use this argument if the -CCsecurityManager parameter is set to enable. When you enable this parameter, the java.security.policy system property is set. The default is <install_root>/properties/client.policy.

-CCD

Use this option to have the WebSphere Application Server set the specified system property during initialization. Do not use the = character after the -CCD. For example: -CCDcom.ibm.test.property=testvalue. You can specify multiple -CCD parameters. The general format of this parameter is -CCD<property key>=<property value>.

-CCexitVM

Use this option to have the WebSphere Application Server call System.exit() after the client application completes. The default is false.

-CCdumpJavaNameSpace

Prints out the Java portion of the WebSphere Application Server Java Naming and Directory Interface (JNDI) name space. The true value uses the short format which prints out the binding name and the type of the object bound at that location. The long value uses the long format which prints out the binding name, bound object type, local object, type, and string representation of the local object, for example: IORs, and string values. The default value is false.

The following examples demonstrate correct syntax.

On the Windows operating system:

```
launchClient c:\earfiles\myapp.ear -CCBootstrapHost=myWASServer  
-CCverbose=true app_parm1 app_parm2
```

On the UNIX operating system:

```
./launchClient.sh /usr/earfiles/myapp.ear  
-CCBootstrapHost=myWASServer -CCverbose=true app_parm1 app_parm2
```

Specifying the directory for an expanded EAR file

Each time `launchClient` is called, it extracts the EAR file to a random directory name in the temporary directory on your hard drive. Then it sets up the thread `ClassLoader` to use the extracted EAR file directory and JAR files included in the `Manifest.mf` client JAR file. In a normal J2EE Java client, these files are automatically cleaned up after the application exits. This cleanup occurs when the client container shutdown hook is called. To avoid extracting the EAR file (and removing the temporary directory) each time `launchClient` is called, complete the following steps:

1. Specify a directory to extract the EAR file by setting the `com.ibm.websphere.client.applicationclient.archivedir` Java system property. If the directory does not exist or is empty, the EAR file is extracted normally. If the EAR file was previously extracted, the `launchClient` tool reuses the directory.
2. Delete the directory before running the `launchClient` tool again, if you need to update your EAR file. When you call the `launchClient` command, it extracts the new EAR file to the directory. If you do not delete the directory or change the system property value to point to a different directory, `launchClient` reuses the currently extracted EAR file, and does not use your changed EAR file.

Note: When specifying the `com.ibm.websphere.client.applicationclient.archivedir` property, make sure that the directory you specify is unique for each EAR file you use. For example, do not point `MyEar1.ear` and `MyEar2.ear` files to the same directory.

Application client troubleshooting tips

This section provides some debugging tips for resolving common J2EE application client problems. To use this troubleshooting guide, review the trace entries for one of the J2EE application client exceptions, and then locate the exception in the guide. Some of the errors in the guide are samples; the actual error you receive can be slightly different than what is shown here. Also, it can be useful to rerun the `launchClient` command specifying the `-CCverbose=true` option. This option provides additional information when the J2EE application client run time is initializing

Error: `java.lang.NoClassDefFoundError`

Explanation

This exception is thrown when Java code cannot load the specified class.

Possible causes

- Invalid or non-existent class
- Class path problem
- Manifest problem

Recommended response

Check to determine if the specified class exists in a JAR file within your EAR file. If it does, make sure the path for the class is correct. For example, if you get the exception (shown here on multiple lines for publication):

```
java.lang.NoClassDefFoundError:  
  WebSphereSamples.HelloEJB.HelloHome
```

ensure the class HelloHome exists in one of the JAR files in your EAR file. If it exists, ensure the path for the class is WebSphereSamples.HelloEJB.

If both the class and path are correct, then it is a classpath issue. Most likely, you do not have the failing class JAR file specified in the client JAR file manifest. To verify this situation, perform the following steps:

1. Open your EAR file with the Application Assembly Tool and click on the Application Client.
2. Add the names of the other JAR files in the EAR file to the Classpath field.

This exception is generally caused by a missing EJB module name from the Classpath field.

If you have multiple JAR files to enter in the Classpath field, be sure to separate the JAR names with spaces.

If you still have the problem, you have a situation where a class is loaded from the file system instead of the EAR file. This is a very difficult situation to debug because the offending class is not the one specified in the exception. Instead, another class is loaded from the file system before the one specified in the exception. To correct this problem, review the classpaths specified with the `-CClasspath` option and the classpaths configured with the Application Client Resource Configuration Tool, or the Client Container Resource Configuration Scripting tool for z/OS. Look for classes that also exist in the EAR file. You must resolve the situation where one of the classes is found on the file system instead of in the `.ear` file. Remove entries from the classpaths, or include the `.jar` files and classes in the `.ear` file instead of referencing them from the file system.

If you use the `-CClasspath` parameter or resource classpaths in the Application Client Resource Configuration Tool, or the Client Container Resource Configuration Scripting tool for z/OS, and you have configured multiple JAR files or classes, verify they are separated with the correct character for your operating system. Unlike the classpath field in the Application Assembly Tool, these classpath fields use platform-specific separator characters, usually a colon (on UNIX platforms) or a semi-colon (on

Error: com.ibm.websphere.naming.CannotInstantiateObjectException: Exception occurred while attempting to get an instance of the object for the specified reference object. [Root exception is javax.naming.NameNotFoundException: xxxxxxxxxx]

Explanation

This exception occurs when you perform a lookup on an object that is not installed on the host server. Your program can look up the name in the local client Java Naming and Directory Interface (JNDI) name space, but received a NameNotFoundException exception because it is not located on the host server. One typical example is looking up an enterprise bean that is not installed on the host server that you access. This exception might also occur if the JNDI name you configured in your Application Client module does not match the actual JNDI name of the resource on the host server.

Possible causes

- Incorrect host server invoked
- Resource is not defined
- Resource is not installed
- Application server is not started
- Invalid JNDI configuration

Recommended response

If you are accessing the wrong host server, run the **launchClient** command again with the **-CCBootstrapHost** parameter specifying the correct host server name. If you are accessing the correct host server, use the WebSphere **dumpnamespace** command line tool to see a listing of the host server JNDI name space. If you do not see the failing object name, the resource is either not installed on the host server or the appropriate application server is not started. If you determine the resource is already installed and started, your JNDI name in your client application does not match the global JNDI name on the host server. Use the Application Assembly Tool to compare the JNDI bindings value of the failing object name in the client application to the JNDI bindings value of the object in the host server application. They must match.

Error: javax.naming.ServiceUnavailableException: A communication failure occurred while attempting to obtain an initial context using the provider url: "iiop://[invalidhostname]". Make sure that the host and port information is correct and that the server identified by the provider URL is a running name server. If no port number is specified, the default port number 2809 is used. Other possible causes include the network environment or workstation network configuration. Root exception is org.omg.CORBA.INTERNAL: JORB0050E: In Profile.getAddress(), InetAddress.getByName[invalidhostname] threw an UnknownHostException. minor code: 4942F5B6 completed: Maybe

Explanation	This exception occurs when you specify an invalid host server name.
Possible causes	<ul style="list-style-type: none"> • Incorrect host server invoked • Invalid host server name
Recommended response	Run the launchClient command again and specify the correct name of your host server with the -CCBootstrapHost parameter.

Error: javax.naming.CommunicationException: Could not obtain an initial context due to a communication failure. Since no provider URL was specified, either the bootstrap host and port of an existing ORB was used, or a new ORB instance was created and initialized with the default bootstrap host of "localhost" and the default bootstrap port of 2809. Make sure the ORB bootstrap host and port resolve to a running name server. Root exception is org.omg.CORBA.COMM_FAILURE: WRITE_ERROR_SEND_1 minor code: 49421050 completed: No

Explanation	This exception occurs when you run the launchClient command to a host server that does not have the Application Server started. You also receive this exception when you specify an invalid host server name. This situation might occur if you do not specify a host server name when you run launchClient . The default behavior is for launchClient to run to localhost, because WebSphere Application Server does not know the name of your host server. This default behavior only works when you are running the client on the same computer with WebSphere Application Server is installed.
Possible causes	<ul style="list-style-type: none"> • Incorrect host server invoked • Invalid host server name • Invalid reference to localhost • Application server is not started • Invalid bootstrap port
Recommended response	If you are not running to the correct host server, run the launchClient command again and specify the name of your host server with the -CCBootstrapHost parameter. Otherwise, start the Application Server on the host server and run the launchClient command again.

Error: javax.naming.NameNotFoundException: Name comp/env/ejb not found in context "java:"

Explanation	This exception is thrown when the Java code cannot locate the specified name in the local JNDI name space.
--------------------	--

Possible causes	<ul style="list-style-type: none"> • No binding information for the specified name • Binding information for the specified name is incorrect • Wrong class loader was used to load one of the program classes • A resource reference does not include any client configuration information
Recommended response	<p>Open the EAR file with the Application Assembly Tool and check the bindings for the failing name. Ensure this information is correct. If you are using Resource References, open the EAR file with the Application Client Resource Configuration Tool, and make sure the Resource Reference has client configuration information and the name of the Resource Reference exactly matches the JNDI name of the client configuration. If it is correct, you might have a class loader issue.</p>

Error: java.lang.ClassCastException: Unable to load class: org.omg.stub.WebSphereSamples.HelloEJB._HelloHome_Stub at com.ibm.rmi.javax.rmi.PortableRemoteObject.narrow (portableRemoteObject.java:269)

Explanation	<p>This exception occurs when the application program attempts to narrow to the EJB home class and the class loaders cannot find the EJB client side bindings.</p>
Possible causes	<ul style="list-style-type: none"> • The files, *_Stub.class and _Tie.class, are not in the EJB .jar file • Class loader could not find the classes
Recommended response	<p>Look at the EJB .jar file located in the .ear file and verify the class contains the EJB client side bindings. These are class files whose names end in _Stub and _Tie. If these files are not present, then use the Application Assembly Tool to generate the binding classes. For more information, see article Generating deployment code for modules. If the binding classes are in the EJB .jar file, then you might have a class loader issue.</p>

Error: WSCL0210E: The Enterprise archive file [EAR file name] could not be found. com.ibm.websphere.client.applicationclient.ClientContainerException: com.ibm.etools.archive.exception.OpenFailureException

Explanation	<p>This error occurs when the application client run time cannot read the Enterprise Archive (EAR) file.</p>
Possible causes	<p>The most likely cause of this error is that the system cannot find the EAR file cannot be found in the path specified on the launchClient command.</p>

Recommended response

Verify that the path and file name specified on the **launchclient** command are correct. If you are running on the Windows operating system and the path and file name are correct, use a short version of the path and file name (8 character file name and 3 character extension).

The launchClient command appears to hang and does not return to the command line when the client application has finished.**Explanation**

When running your application client using the **launchClient** command the WebSphere Application Server run time might need to display the security login dialog. To display this dialog the WebSphere Application Server run time creates an Abstract Window Toolkit (AWT) thread. When your application returns from its main method to the application client run time, the application client run time attempts to return to the operating system and end the Java Virtual Machine code. However, since there is an AWT thread, the Java Virtual Machine code will not end until `System.exit` is called.

Possible causes

The Java Virtual Machine code does not end because there is an AWT thread. Java code requires that `System.exit()` be called to end AWT threads.

Recommended response

- Modify your application to call `System.exit(0)` as the last statement.
- Use the `-CCexitVM=true` parameter when you call the **launchClient** command.

Chapter 7. Web Services Invocation Framework (WSIF): Enabling Web services

The Web Services Invocation Framework (WSIF) is a WSDL-oriented Java API. You use this API to invoke Web services dynamically, regardless of the service implementation format (for example enterprise bean (EJB)) or the service access mechanism (for example Java Messaging Service (JMS)).

Using WSIF, you can move away from the usual Web services programming model of working directly with the SOAP APIs, towards a model where you interact with representations of the services. So you can work with the same programming model regardless of how the service is implemented and accessed.

If you want to know more about the issues that WSIF addresses, see [Goals of WSIF](#).

If you want to know how WSIF addresses these issues, see [An overview of WSIF](#).

To use WSIF, see the following topics:

- [Using WSIF to invoke Web services](#).
- [WSIF system management and administration](#).
- [WSIF API](#).

For more information about working with WSIF, visit the Internet sites listed in [WSIF: Resources for Learning](#).

Goals of WSIF

SOAP bindings for Web services are part of the WSDL specification. So when most developers think of using a Web service, they immediately think of assembling a SOAP message and sending it across the network to the service endpoint, using a SOAP client API. For example: using Apache SOAP the client creates and populates a Call object that encapsulates the service endpoint, the identification of the SOAP operation to invoke, the parameters to send, and so on.

While this process works for SOAP, it is limited in its use as a general model for invoking Web services for the following reasons:

- Web services are more than just SOAP services.
- Tying client code to a particular protocol implementation is restricting.
- Incorporating new bindings into client code is hard.
- Multiple bindings can be used in flexible ways.
- A freer Web services environment enables intermediaries.

The goals of the Web Services Invocation Framework (WSIF) are therefore:

- To give a binding-independent mechanism for Web service invocation.
- To free client code from the complexities of any particular protocol used to access a Web service.
- To enable dynamic selection between multiple bindings to a Web service.
- To help the development of Web service intermediaries.

WSIF - Web services are more than just SOAP services

You can deploy as a Web service any application that has a WSDL-based description of its functional aspects and access protocols. If you are using the Java 2 platform, Enterprise Edition (J2EE) environment, then the application is available over multiple transports and protocols.

For example, you can take a database-stored procedure, expose it as a stateless session bean, then deploy it into a SOAP router as a SOAP service. At each stage, the fundamental service is the same. All that changes is the access mechanism: from Java Database Connectivity (JDBC) to Remote Method Invocation over Internet Inter-Orb Protocol (RMI-IIOP) and then to SOAP.

The WSDL specification defines a SOAP binding for Web services, but you can add binding extensions to the WSDL so that, for example, you can offer an enterprise bean as a Web service using RMI-IIOP as the access protocol. You can even treat a single Java class as a Web service, with in-thread Java method invocations as the access protocol. With this broader definition of a Web service, you need a binding-independent mechanism for service invocation.

WSIF - Tying client code to a particular protocol implementation is restricting

If your client code is tightly bound to a client library for a particular protocol implementation, it can become hard to maintain.

For example, if you move from Apache SOAP to Java Messaging Service (JMS) or enterprise bean, the process can take a lot of time and effort. To avoid these problems, you need a protocol implementation-independent mechanism for service invocation.

WSIF - Incorporating new bindings into client code is hard

As is explained in Web services are not just SOAP services, if you want to make an application that uses a custom protocol work as a Web service, you can add extensibility elements to WSDL to define the new bindings. But in practice, achieving this capability is hard. For example you have to design the client APIs to use this protocol. If your application uses just the abstract interface of the Web service, you have to write tools to generate the stubs that enable an abstraction layer. These tasks can take a lot of time and effort. What you need is a service invocation mechanism that allows you to update existing bindings, and to add new bindings.

WSIF - Multiple bindings can be used in flexible ways

Imagine that you have successfully deployed an application that uses a Web service which offers multiple bindings. For example, imagine that you have a SOAP binding for the service and a local Java binding that lets you treat the local service implementation (a Java class) as a Web service.

The local Java binding for the service can only be used if the client is deployed in the same environment as the service. In this case, it is more efficient to communicate with the service by making direct Java calls than by using the SOAP binding.

If your clients could switch the actual binding used based on run-time information, they could choose the most efficient available binding for each situation. To take advantage of Web services that offer multiple bindings, you need a service

invocation mechanism that can switch between the available service bindings at run-time, without having to generate or recompile a stub.

WSIF - Enabling a freer Web services environment promotes intermediaries

Web services offer application integrators a loosely-coupled paradigm. In such environments, intermediaries can be very powerful.

Intermediaries are applications that intercept the messages that flow between a service requester and a target Web service, and perform some mediating task (for example logging, high-availability or transformation) before passing on the message. They can be as small as a simple Web service, or as large as the Web services gateway. The Web Services Invocation Framework (WSIF) is designed to make building intermediaries both possible and simple. Using WSIF, intermediaries can add value to the service invocation without needing transport-specific programming.

An overview of WSIF

The Web Services Invocation Framework (WSIF) provides a Java API for invoking Web services, independent of the format of the service or the transport protocol through which it is invoked. This framework addresses all of the issues identified in the goals of WSIF.

WSIF provides the following features:

- An API that provides binding-independent access to any Web service.
- A close relationship with WSDL, so it can invoke any service that you can describe in WSDL.
- A stubless and completely dynamic invocation of a Web service.
- The capability to plug a new or updated implementation of a binding into WSIF at run-time.
- The option to defer the choice of a binding until run-time.

WSIF is designed to work both in an unmanaged environment (stand-alone) and inside a managed container. You can use the Java Naming and Directory Interface (JNDI) to find the WSIF service, or you can use the location described in the WSDL.

For more conceptual information about WSIF and WSDL, see the following topics:

- WSIF and WSDL
- WSIF architecture
- Using WSIF with Web services that offer multiple bindings
- WSIF usage scenarios
- Dynamic invocation

WSIF architecture

The Web Services Invocation Framework (WSIF) architecture is shown in the

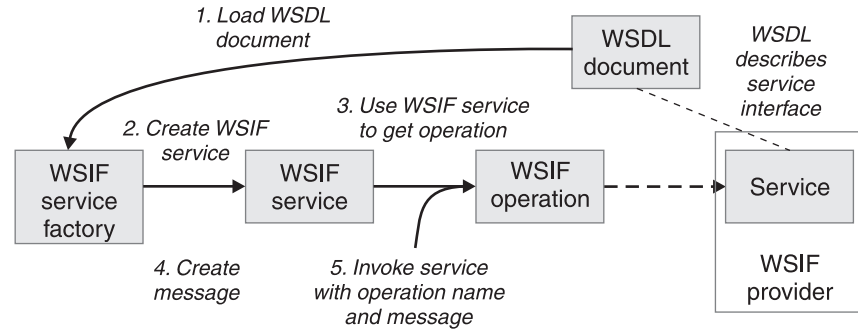


figure.

The components of this architecture include:

WSDL document

The Web service WSDL document contains the location of the Web service. The binding document defines the protocol and format for operations and messages defined by a particular portType.

WSIF service

The WSIFService is responsible for generating an instance of WSIFOperation to be used for a particular invocation of a service operation. For more information, see Finding a port factory or service

WSIF operation

The run-time representation of an operation, called WSIFOperation is responsible for invoking a service based on a particular binding. For more information, see WSIF API reference: Using ports.

WSIF provider

A WSIF provider is an implementation of a WSDL binding that can run a WSDL operation through a binding-specific protocol. WSIF includes SOAP providers, JMS providers, Java providers and EJB providers. For more information see Using the WSIF providers.

Using WSIF with Web services that offer multiple bindings

Using WSIF, a client application can choose dynamically the optimal binding to use for invoking Web service operations.

For example, a Web service might offer a SOAP binding, and also a local Java binding so that you can treat the local service implementation (a Java class) as a Web service. If a client application is deployed in the same environment as the service, then this client can use the local Java binding for the service. This provides more efficient communication between the client and the service by making direct Java calls rather than indirect calls using the SOAP binding.

For more information on how to configure a client to dynamically select between multiple bindings, see Developing a WSIF service.

WSIF and WSDL

WSDL is the acronym for Web Services Description Language.

In WSDL a service is defined in three distinct sections:

- The **portType**. This section defines the abstract interface offered by the service. A portType defines a set of *operations*. Each operation can be In-Out (request-response), In-Only, Out-Only and Out-In (Solicit-Response). Each

operation defines the input and/or output *messages*. A message is defined as a set of *parts*, and each part has a schema-defined type.

- The **binding**. This section defines how to map between the abstract portType and a real service format and protocol. For example the SOAP binding defines the encoding style, the SOAPAction header, the namespace of the body (the targetURI), and so on.
- The **port**. This section defines the actual location (endpoint) of the available service. For example, the HTTP Web address at which a SOAP service is available.

Currently in WSDL, each port has one and only one binding, and each binding has a single portType. But (more importantly) each Service (portType) can have multiple ports, each of which represents an alternative location and binding for accessing that service.

The Web Services Invocation Framework (WSIF) follows the semantics of WSDL as much as possible:

- The WSIF dynamic invocation API directly exposes run-time equivalents of the model from WSDL. For example, invocation of an operation involves executing an operation with an input message.
- WSDL has extension points that support the addition of new ports and bindings so that WSDL can describe new systems. The equivalent concept in WSIF is a provider, that enables WSIF to understand a class of extensions and thereby to support a new service implementation type.

As a metadata-based invocation framework, WSIF follows the design of the metadata. As WSDL is extended, WSIF is updated to follow.

The implicit and primary type system of WSIF is XML schema. WSIF supports invocation using dynamic proxies, which in turn support Java type systems, but when you use the WSIFMessage interface it is your responsibility to populate WSIFMessage objects with data based on the XML schema types as defined in the WSDL document. You should define your object types by a canonical and fixed mapping from schema types into the run-time.

For more information on WSDL, see WSIF: Resources for learning.

WSIF usage scenarios

This topic describes two brief scenarios that illustrate the role WSIF plays in the emerging Web services environment.

Scenario: Redevelopment and redeployment

When you first implement a Web service, you create a simple prototype. When you want to move a prototype Web service into production, you often need to redevelop and redeploy it.

The Web Services Invocation Framework (WSIF) uses the same API calls irrespective of the underlying technologies. So if you use WSIF:

- You can reimplement and redeploy your services without changing the client code.
- You can use existing reliable and high-performance infrastructures like Remote Method Invocation over Internet Inter-Orb Protocol (RMI-IIOP) and Java Messaging Service (JMS) without sacrificing the location-independence that the Web service model offers.

Scenario: Service Flow composition

A service flow typically invokes a Web service, then passes the response from one Web service to the next Web service, perhaps performing some transformation in the middle.

There are two key aspects to this flow that WSIF provides:

- A representation of the service invocation based on the metadata in WSDL.
- The ability to build invocations based solely on the portType, which can therefore be used in any implementation.

For example, imagine that you build a meta-service that uses a number of services to build a process. Initially, several of those services are simple Java bean prototypes that are written and exposed through SOAP, but you plan to reimplement some of them as EJB components, and to out-source others.

If you use SOAP, it ties up multiple threads for every onward invocation, because they pass through the Web server and servlet engine and on to the SOAP router. If you use WSIF to call the beans directly, you get much better performance compared to SOAP and you don't lose access or location transparency. Using WSIF, you can replace the Java bean implementations with EJB implementations without changing the client code. To move some of the Web services from local implementations to external SOAP services, you just update the WSDL.

Dynamic invocation

For the Web Services Invocation Framework (WSIF), dynamic invocation means providing the following levels of support when invoking Web services:

1. Support for WSDL extensions and bindings that were not known at build time.
2. Support for Web services that were not known at build time.

WSIF supports (1) through the use of providers.

The providers support (2) by using the WSDL description to access the target service.

Using WSIF to invoke Web services

You invoke a Web service dynamically by using the WSIF API directly.

You only specify the location of the WSDL file for the service, the name of the operation to invoke, and any operation arguments. All the information needed to access the Web service (the abstract interface, the binding, and the service endpoint) is available through the WSDL.

This kind of invocation does not require stub classes and does not need a separate compilation cycle.

More information on using the Web Services Invocation Framework (WSIF) to invoke Web services is given in the following topics:

- Using the WSIF providers.
- Developing a WSIF service.
- Using complex types.
- Using the Java Naming and Directory Interface (JNDI).
- Interacting with the J2EE container in WebSphere Application Server.
- Running WSIF as a client.

Using the WSIF providers

A Web Services Invocation Framework (WSIF) provider is an implementation of a WSDL binding that can run a WSDL operation through a binding-specific protocol.

Providers implement the interface between the WSIF API and the actual implementation of a service. Providers are pluggable within the WSIF framework, and are registered according to the namespace of the WSDL extension that they implement. Some providers use the Java 2 platform, Enterprise Edition (J2EE) programming model to utilize J2EE services. If a provider is available, but its required class libraries are not, then the provider is disabled.

WebSphere Application Server includes the following WSIF providers:

- SOAP (over HTTP) provider.
- JMS providers (SOAP over JMS, and native JMS).
- Java provider.
- EJB provider.

Note:

-
-

Using the SOAP provider

The SOAP provider allows WSIF stubs and dynamic clients to invoke SOAP services.

The Web Services Invocation Framework (WSIF) SOAP provider supports SOAP 1.1 over HTTP. The SOAP provider uses Apache SOAP 2.3 for parsing and creating SOAP messages, but it is not limited to invoking services from Apache SOAP.

The WSIF SOAP provider supports:

- SOAP-ENC encoding.
- **5.0+** **5.0.1** Remote Procedure Call (RPC) style SOAP messages.

The SOAP provider is not transactional.

If you have a Web service that you expect multiple clients to use connecting over SOAP, then before you deploy the service you must set up your application deployment descriptor file `dds.xml` to handle multiple connections correctly. For more information see WSIF troubleshooting tips.

For an example of the sort of code changes that need to be made in the WSDL file for a SOAP provider, see the following topics:

- The SOAP over JMS provider - writing the WSDL extension.

Using the JMS providers

The JMS providers enable a WSIF service to be invoked through JMS.

The Java Messaging Service (JMS) is an API for transport technology. The mapping to a JMS destination is defined during deployment and maintained by the container.

The JMS Destination Web service endpoint can be realized in any of the following ways:

- The JMS destination for the queue can be the Web service implementation.

- The JMS destination can be (but is not required to be) associated with a Message Driven Bean by the EJB container, thereby allowing the Message Driven Bean to be the Web service implementation.
- (For SOAP over JMS) The JMS Destination can unwrap the JMS message and route the SOAP message to a stateless session bean Web service implementation.

The JMS Destination must respect the interaction model expected by the client and defined by the WSDL. It must return a response if one is required.

When the JMS Destination creates the JMS response message the following rules must be adhered to:

- The response message must be sent to the **JMSReplyTo** from the incoming request.
- The **JMSCorrelationID** of the response message must be set to the value of the **JMSMessageID** from the request message.
- The response must be sent with a **deliveryMode** equal to the **JMSDeliveryMode** of the request message.
- The response must be sent with a **priority** equal to the **JMSPriority** of the request message.
- The **timetolive/JMSExpiration** must be set a value equal to the **JMSExpiration** of the request message.

The client does not see any of these headers. The container receives the JMS message and (for SOAP over JMS) removes the SOAP message to send to the client.

See also the following topics:

- Using the SOAP over JMS provider
- Using the native JMS provider
- The JMS providers - configuring the client and server

Using the SOAP over JMS provider:

For information on working with the Java Messaging Service (JMS) API, see Using the JMS providers.

The SOAP message, including the SOAP envelope, is wrapped with a JMS message and put on the appropriate Queue. The container receives the JMS message and removes the SOAP message to send to the client.

For detailed implementation information, see the following topics:

- The SOAP over JMS provider - writing the WSDL extension
- The JMS providers - configuring the client and server

The SOAP over JMS provider - writing the WSDL extension:

If a SOAP message contains only XML, then it can be carried on the Java Messaging Service (JMS) transport with the JMS message body type **TextMessage**.

The WSDL binding extension for SOAP over JMS varies only slightly from the SOAP over HTTP binding.

Selecting the SOAP over JMS binding

You set the transport attribute of the `<soap:binding>` tag to indicate that JMS is used. If you also set the `style` attribute to `rpc` (Remote Procedure

Call), then the Web Services Invocation Framework (WSIF) assumes that an operation is invoked on the Web service endpoint (shown here on multiple lines for publication):

```
<soap:binding style="rpc"
  transport="http://schemas.xmlsoap.org/soap/jms"/>
```

Setting the JMS address

For SOAP over JMS, the `<wsdl:port>` tag must contain a `<jms:address>` element. This element provides the information required for a client to connect correctly to the Web service using the JMS programming model. Typically, it is the stubs generated to support the SOAP over JMS binding that act as the JMS client. Alternatively, the Web service client can use the JMS programming model directly.

The `<jms:address>` element takes this form:

```
<jms:address
  destinationStyle="queue"
  jmsVendorURI="http://ibm.com/ns/mqseries"?
  initialContextFactory="com.ibm.NamingFactory"?
  jndiProviderURL="iiop://something:9000/wherever"?
  jndiConnectionFactoryName="orange"
  jndiDestinationName="fred"
/>
```

where attributes marked with a question mark (?) are optional.

The optional `jmsVendorURI` attribute is a string that uniquely identifies the JMS implementation. WSIF ignores this URI, which is used by the client developer and perhaps the client implementation to determine if it has access to the correct JMS provider in the client run-time.

The optional attributes `initialContextFactory` and `jndiProviderURL` can only be omitted if the run-time has a default Java Naming and Directory Interface (JNDI) provider configured.

The `jndiConnectionFactoryName` attribute gives the name of a JMS `ConnectionFactory` object, which can be looked up within the JNDI context given by the `jndiContext` attribute

. This `ConnectionFactory` object is used to create a JMS connection to the JMS provider instance that owns the queue. In a simple configuration, the same `ConnectionFactory` object is used by the server message listener and by the clients. However the server and the clients can use different `ConnectionFactory` objects, provided that they all create connections to the same JMS provider instance.

Setting the JMS headers and properties

You use the `<jms:property>` tag to set the JMS headers and properties. This tag maps either a message part, or a literal value, into a JMS property (shown here on multiple lines for publication):

```
<jms:property name="Priority"
  {part="requestPriority" | value="fixedValue"}/>
```

If the `<jms:property>` has a literal value, then it can also be nested within the `<jms:address>` tag:

```
<jms:property name="Priority" value="fixedValue" />
```

This form of the <jms:property> tag is also used in the native JMS binding.

Here is an example of a WSDL that defines a SOAP over JMS binding:

```
<!-- Example: SOAP over JMS Text Message -->

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  name="StockQuoteInterfaceDefinitions"
  targetNamespace="urn:StockQuoteInterface"
  xmlns:tns="urn:StockQuoteInterface"
  xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:message name="GetQuoteInput">
    <part name="symbol" type="xsd:string"/>
  </wsdl:message>
  <wsdl:message name="GetQuoteOutput">
    <part name="value" type="xsd:float"/>
  </wsdl:message>

  <wsdl:portType name="StockQuoteInterface">
    <wsdl:operation name="GetQuote">
      <wsdl:input message="tns:GetQuoteInput"/>
      <wsdl:output message="tns:GetQuoteOutput"/>
    </wsdl:operation>
  </wsdl:portType>

  <wsdl:binding name="StockQuoteSoapJMSBinding"
    type="tns:StockQuoteInterface">
    <soap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/jms"/>
    <wsdl:operation name="GetQuote">
      <soap:operation soapAction="urn:StockQuoteInterface#GetQuote"/>
      <wsdl:input>
        <soap:body use="encoded" namespace="urn:StockQuoteService"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="encoded" namespace="urn:StockQuoteService"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>

  <wsdl:service name="StockQuoteService">
    <wsdl:port name="StockQuoteServicePort"
      binding="sqi:StockQuoteSoapJMSBinding">
      <jms:address destinationStyle="queue"
        jndiConnectionFactoryName="myQCF"
        jndiDestinationName="myQ"
        initialContextFactory="com.ibm.NamingFactory"
        jndiProviderURL="iiop://something:900/" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Using the native JMS provider:

Using the native JMS provider, WSIF clients can treat a JMS destination as a Web service.

For information on working with the Java Messaging Service (JMS) API, see Using the JMS providers.

For detailed implementation information, see the following topics:

- The native JMS provider - writing the WSDL extension
- The JMS providers - configuring the client and server

The native JMS provider - writing the WSDL extension:

The WSDL extensions for the Java Messaging Service (JMS) are identified with the namespace prefix `jms`. For example, `<jms:binding>`.

Operations

The supported operations are either one-way operations (send for JMS point-to-point messaging, or publish for JMS publish and subscribe messaging) or request-response operations (send and receive for JMS point-to-point messaging). The WSDL operations therefore specify either an input message only, or an input and an output message.

Fault messages

Operations that describe message interfaces with a native JMS binding do not have fault messages. No assumptions are made about the message schema or the semantics of message properties, so no distinction can be made between output and fault messages.

Setting the JMS message body type

You use the `<jms:binding>` extension to specify the JMS message body type:

```
<wsdl:binding ... >
  <jms:binding type="messageBodyType" />
  ...
</wsdl:binding>
```

where *messageBodyType* is either `ObjectMessage` or `TextMessage`.

Specifying the parts to use for the input and output messages

For JMS text messages and JMS object messages created from one or more WSDL message parts, you use the `<jms:input>` and `<jms:output>` extensions to specify the message parts to use for the JMS messages:

```
<wsdl:input ... >
  <jms:input parts="part1 part2 ..." />
</wsdl:input>

<wsdl:output ... >
  <jms:output parts="part1 part2 ..." />
</wsdl:output>
```

In the next example, the WSDL message has just one part that contains the complete message body. This message body might result from a mapping of some other representation (see mapping data types).

```
<wsdl:input ... >
  <jms:input parts="part1" />
</wsdl:input>
```

If no parts are defined, then all the message parts are used.

Mapping data types

You use the `<format>` extensions to map data types:

```
<wsdl:binding ... >
  <jms:binding type="..." />

  <format:typeMapping encoding="Java" style="Java">
```

```

        <format:typeMap typeName="..." formatType="targetType"/>
    </format:typemapping>
    ...
</wsdl:binding>

```

The value of *targetType* is dependent on the JMS message body type. For JMS object messages, the target data type implements the `java.io.Serializable` class. For JMS text messages, the target data type is always `java.lang.String`.

The `<format>` extensions are also used in other bindings that deal with Java interfaces.

Setting the JMS headers and properties

JMS does not make assumptions about message headers. For example, if the JMS provider is MQSeries then each JMS message carries an RFH2 header. However you can access data in this message header indirectly, by getting and setting JMS message properties.

When you want your application to pass a property into the Web Services Invocation Framework (WSIF) as a part on the WSIF message, you use a `<jms:property>` tag. When you want to hard code an actual property value into the WSDL, you use a `<jms:propertyValue>` tag. The `<jms:propertyValue>` tag contains a specification of a literal value and its associated XML schema type.

You can specify `<jms:property>` and `<jms:propertyValue>` extensions within the `<wsdl:input>` tag in the binding operation, and also within the `<jms:address>` tag. For the `<wsdl:output>` tag in the binding operation, you can only specify the `<jms:property>` extension. Property values that are set in the `<jms:property>` tag take precedence over values set in the `<jms:propertyValue>` tag, and property values that are set in the binding operation (in the `<input>` and `<output>` tags) take precedence over values set in the `<jms:address>` tag.

Here is an example of the `<jms:property>` and `<jms:propertyValue>` tags nested within the `<input>` and `<output>` tags:

```

<wsdl:input ... >
    <jms:property name="propertyName" part="partName" />
    <jms:propertyValue name="propertyName"
        type="xsdType" value="actualValue" />
</wsdl:input>
<wsdl:output ... >
    <jms:property name="propertyName" part="partName" />
</wsdl:output>

```

where *propertyName* identifies the JMS property that is associated with the header field, and *partName* identifies the message part that is associated with the property.

The JMS property identified by *propertyName* can be user-defined, or it can be one of the following predefined JMS message header fields:

Value	Java type
JMSMessageId	java.lang.String
JMSTimeStamp	long
JMSCorrelationId	byte [] or java.lang.String
JMSReplyTo	javax.jms.Destination
JMSDestination	javax.jms.Destination
JMSDeliveryMode	int
JMSRedelivered	boolean
JMSType	java.lang.String
JMSExpiration	long

See the JMS specification for restrictions that apply when setting JMS header field values. Attempts to set restricted values are ignored.

For application-defined JMS message properties, the Java types used in the native JMS binding implementation (used for calls to the corresponding JMS methods) are derived from the XML schema type in the abstract interface (<wsdl:part> tag), and from the type mapping information in the format binding (<format:typemap> tag).

Handling transactions

Independent of other JMS properties, the asynchronous processing of request-response operations has implications for callers running in a transaction scope. The send request part and the receive response part are separated into two transactions, because the send needs to be committed in order for the request message to become visible.

So implementations that process WSDL for asynchronous request-response operations (such as WSIF) must take the following additional actions:

- They must ensure that the send request transaction returns a correlation ID to the user, and provides a **callback** with which users can pass in the response message to process the receive response transaction.
- They might implement their own response message “listener” in order to recognize the arrival of response messages, and to manage the correlation to the request message.

The JMS text message contains a **java.lang.String**. In this example, the WSDL message contains only one part that represents the whole message body:

```
<!-- Example 1: JMS Text Message -->
<wsdl:definitions ... >

  <!-- simple or complex types for input and output message -->
  <wsdl:types> ... </wsdl:types>

  <wsdl:message name="JmsOperationRequest"> ... </wsdl:message>
  <wsdl:message name="JmsOperationResponse"> ... </wsdl:message>

  <wsdl:portType name="JmsPortType">
    <wsdl:operation name="JmsOperation">
      <wsdl:input name="Request"
        message="tns:JmsOperationRequest"/>
      <wsdl:output name="Response"
        message="tns:JmsOperationResponse"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

```

</wsdl:portType>

<wsdl:binding name="JmsBinding" type="JmsPortType">
  <jms:binding type="TextMessage" />

  <format:typemapping style="Java" encoding="Java">
    <format:typemap name="xsd:String" formatType="String" />
  </format:typemapping>

  <wsdl:operation name="JmsOperation">
    <wsdl:input message="JmsOperationRequest">
      <jms:input parts="requestMessageBody" />
    </wsdl:input>
    <wsdl:output message="JmsOperationResponse">
      <jms:output parts="responseMessageBody" />
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>

<wsdl:service name="JmsService">
  <wsdl:port name="JmsPort" binding="JmsBinding">
    <jms:address destinationStyle="queue"
      jndiConnectionFactoryName="myQCF"
      jndiDestinationName="myDestination" />
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

As an extension to the previous JMS message example, the following example WSDL describes a request-response operation in which specific JMS property values of the request and response message are set for the request message and retrieved from the response message.

The JMS properties in the request message are set according to the values in the input message. Likewise, selected JMS properties of the response message are copied to the corresponding values of the output message. The direction of the mapping is determined by the appearance of the <jms:property> tag in the input or output section, respectively.

<!-- Example 2: JMS Message with JMS Properties -->

```

<wsdl:definitions ... >

  <!-- simple or complex types for input and output message -->
  <wsdl:types> ... </wsdl:types>

  <wsdl:message name="JmsOperationRequest">
    <wsdl:part name="myInt" type="xsd:int"/>
    ...
  </wsdl:message>

  <wsdl:message name="JmsOperationResponse">
    <wsdl:part name="myString" type="xsd:String"/>
    ...
  </wsdl:message>

  <wsdl:portType name="JmsPortType">
    <wsdl:operation name="JmsOperation">
      <wsdl:input name="Request"
        message="tns:JmsOperationRequest"/>
      <wsdl:output name="Response"
        message="tns:JmsOperationResponse"/>
    </wsdl:operation>
  </wsdl:portType>

```

```

<wsdl:binding name="JmsBinding" type="JmsPortType">
  <!-- the JMS message type may be any of the above -->
  <jms:binding type="..." />

  <format:typemapping style="Java" encoding="Java">
    <format:typemap name="xsd:int" formatType="int" />
    ...
  </format:typemapping>

  <wsdl:operation name="JmsOperation">
    <wsdl:input message="JmsOperationRequest">
      <jms:property message="tns:JmsOperationRequest" parts="myInt" />
      <jms:propertyValue name="myLiteralString"
        type="xsd:string" value="Hello World" />
      ...
    </wsdl:input>
    <wsdl:output message="JmsOperationResponse">
      <jms:property message="tns:JmsOperationResponse" parts="myString" />
      ...
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>

<wsdl:service name="JmsService">
  <wsdl:port name="JmsPort" binding="JmsBinding">
    <jms:address destinationStyle="queue"
      jndiConnectionFactoryName="myQCF"
      jndiDestinationName="myDestination"/>
  </wsdl:port>
</wsdl:service>

</wsdl:definitions>

```

The JMS providers - configuring the client and server:

To enable a service to be invoked through the Java Messaging Service (JMS) by a Web Services Invocation Framework (WSIF) client application, complete the following steps:

1. Create a JMS queue to which WSIF can send the request message. You can use MQSeries Explorer, or an equivalent tool from your JMS implementation package.

Note:

2. Create a Java Naming and Directory Interface (JNDI) definition for that JMS queue, and a JNDI queue connection factory.

Note: If you are using MQSeries, you can create a definition using the `jmsadmin` tool.

3. Put the JNDI names of the queue and queue connection factory, as well as your JNDI configuration, in the WSDL file.

You should also understand the specific ways in which WSIF interacts with JMS:

- Only input JMS properties are supported.
- WSIF needs two queues when invoking an operation: one for the request message and one for the reply. The `replyTo` queue is by default a temporary queue, which WSIF creates on behalf of the application. You can specify a permanent queue by setting the `JMSReplyTo` property to the JNDI name of a queue.
- WSIF uses the default values for properties set by the JMS implementation. However in MQSeries and in some other JMS implementations, messages are persistent by default, and the default temporary queue is of type temporary

dynamic and cannot have persistent messages written to it. Therefore your JMS listener can fail to write a persistent response message to the temporary replyTo queue.

Note: If you are using MQSeries, you need to create a temporary model queue that is of type permanent dynamic, then pass this model as the tempmodel of your queue connection factory. This will ensure that persistent messages are written to a temporary replyTo queue that is of type permanent dynamic.

Using the Java provider

Using the WSIF Java provider, WSIF can invoke Java code.

This means that, in a thin-client environment such as a Java Virtual Machine (JVM) or Tomcat test run-time, you can define shortcuts to local Java programs.

The Web Services Invocation Framework (WSIF) Java provider is not intended for use in a Java 2 platform, Enterprise Edition (J2EE) environment. There is a difference between a client using the WSIF Java provider to invoke a Java component, and implementing a Web service as a Java component on the server side.

The Java binding exploits the format binding for type mapping. Using the format binding, your WSDL can define the mapping between XML Schema types and Java types.

The Java provider requires that the targeted Java classes reside in the class path of the client. The Java method is invoked synchronously, in-process, in-thread, with the current thread and Object Request Broker (ORB) contexts.

The Java provider is not transactional.

For examples of the code changes that need to be made in the WSDL file, see The Java provider - writing the WSDL extension.

The Java provider - writing the WSDL extension:

The Java provider supports the invocation of a method on a local Java object.

To use the Java provider, you need the following binding specified in the WSDL:

```
<!-- Java binding -->
<binding .... >
  <java:binding />
  <format:typeMapping style="Java" encoding="Java"/>?
    <format:typeMap name="qname" formatType="nmtoken"/>*
  </format:typeMapping>
  <operation>*
    <java:operation
      methodName="nmtoken"
      parameterOrder="nmtoken"
      returnPart="nmtoken"?
      methodType="instance|constructor" />
    <input name="nmtoken"? />?
    <output name="nmtoken"? />?
    <fault name="nmtoken"? />?
  </operation>
</binding>
```

In this example:

- A question mark (?) means optional, and an asterisk (*) means 0 or more.
- The name attribute of the <format:typeMap> element is a qualified name of a simple or complex type used by one of the Java operations.
- The formatType attribute of the <format:typeMap> element is the fully qualified class name for the Java class to which the element specified by name maps.
- The methodName attribute of the <java:operation> element is the name of the method on the Java object that is called by the operation.
- The parameterOrder attribute of the <java:operation> element contains a white space-separated list of part names that define the order in which they are passed to the Java object method.
- The methodType attribute of the <java:operation> element must be set to either instance or constructor. The value specifies whether the method that is invoked on the object is an instance method or a constructor for the object.

In the next example, the className attribute of the <java:address> element specifies the fully qualified class name of the object containing the method to invoke:

```
<service ... >
  <port>*
    <java:address
      className="nmtoken"/>
  </port>
</service>
```

Using the EJB provider

Using the EJB provider, WSIF clients can invoke enterprise beans.

The EJB client JAR file must be available in the client run-time with the current provider. The enterprise bean is invoked using normal EJB invocation methods, using Remote Method Invocation over Internet Inter-Orb Protocol (RMI-IIOP), with the current security and transaction contexts. If the EJB provider is invoked within a transaction, the transaction is passed to the onward service and the standard EJB transaction attribute applies.

If there are multiple implementations of the service, it is up to the service providers to make sure that every implementation offers the same semantics. For example, in the case of transactions, the bean deployer must specify TX_REQUIRES_NEW to force a new transaction.

For examples of the code changes that need to be made in the WSDL file, see The EJB provider - writing the WSDL.

The EJB provider - writing the WSDL extension:

The EJB provider supports the invocation of an enterprise bean through Remote Method Invocation over Internet Inter-Orb Protocol (RMI-IIOP).

To use the EJB provider, you need the following binding specified in the WSDL:

```
<!-- EJB binding -->
<binding .... >
  <ejb:binding />
  <format:typeMapping style="Java" encoding="Java"/>?
    <format:typeMap name="qname" formatType="nmtoken"/>*
  </format:typeMapping>
  <operation>*
    <ejb:operation
      methodName="nmtoken"
      parameterOrder="nmtoken"
```

```

        returnPart="nmtoken"?
        interface="remote|home" />
    <input name="nmtoken"? />?
    <output name="nmtoken"? />?
    <fault name="nmtoken"? />?
</operation>
</binding>

```

In this example:

- A question mark (?) means optional, and an asterisk (*) means 0 or more.
- The name attribute of the <format:typeMap> element is a qualified name of a simple or complex type used by one of the EJB operations.
- The formatType attribute of the <format:typeMap> element is the fully qualified class name for the Java class to which the element specified by name maps.
- The methodName attribute of the <ejb:operation> element is the name of the method on the enterprise bean that is called by the operation.
- The parameterOrder attribute of the <ejb:operation> element contains a white space-separated list of part names that define the order in which they are passed to the EJB method.
- The interface attribute of the <ejb:operation> element must be set to either remote or home. The value specifies the interface of the enterprise bean on which the method named by the methodName attribute is accessible.

In the next example:

- The className attribute of the <ejb:address> element specifies the fully qualified class name of the home interface class of the enterprise bean.
- The jndiName attribute of the <ejb:address> element specifies the full Java Naming and Directory Interface (JNDI) name that is used to look up the enterprise bean.
- The initialContextFactory attribute of the <ejb:address> element is optional and specifies the initial context factory class.
- The jndiProviderURL attribute of the <ejb:address> element is optional and specifies the JNDI provider Web address.

```

<service ... >
  <port>*
    <ejb:address
      className="nmtoken"
      jndiName="nmtoken"
      initialContextFactory="nmtoken" ?
      jndiProviderURL="nmtoken" ? />
  </port>
</service>

```

Developing a WSIF service

A Web Services Invocation Framework (WSIF) service is a Web service that uses WSIF.

To develop a WSIF service, develop the Web service (or use an existing Web service), then develop the WSIF client based on the WSDL document for that Web service.

There are also two pre-built WSIF Samples available for download from the Samples Central page of the IBM WebSphere Developer Domain Web site:

- The Address Book Sample.
- The Stock Quote Sample.

For more information on using the pre-built Samples, see the documentation that is included in the download package.

To develop a WSIF service, complete the following steps:

1. Develop the Web service.

Use Web services tools to discover, create, and publish the Web service. You can develop Java bean, enterprise bean, and URL Web services. You can use Web service tools to create skeleton Java code and a sample application from a WSDL document. For example, an enterprise bean can be offered as a Web service, using Remote Method Invocation over Internet Inter-Orb Protocol (RMI-IIOP) as the access protocol. Or you can use a Java class as a Web service, with native Java invocations as the access protocol.

You can use the WebSphere Studio Application Developer to create a Web service from a Java application, as described in its StockQuote service tutorial. The Java application that you use in this scenario returns the last trading price from the Internet Web site www.xmltoday.com, given a stock symbol. Using the Web Service wizard, you generate a binding WSDL document named `StockQuoteService-binding.wsdl` and a service WSDL document named `StockQuoteService-service.wsdl` from the `StockQuoteService.java` bean. You then deploy the Web service to a Web server, generate a client proxy to the Web service, and generate a sample application that accesses the `StockQuoteService` through the client proxy. You test the `StockQuote` Web service, publish it using the IBM UDDI Explorer, and then discover the `StockQuote` Web service in the IBM UDDI Test Registry.

2. Develop the WSIF client. The information you need to develop a WSIF client is given in the following topics:

- [Developing the WSIF client - the Address Book Sample](#) gives example code to show how you define a Web service in WSDL.
- [Using the WSIF providers](#) describes the available providers, and gives example code of how their WSDL extensions are coded.
- [WSIF API](#) defines the main interfaces that your client uses to support the invocation of Web services defined in WSDL.

The `Address Book Sample` is written for synchronous interaction. If you are using a JMS provider, your WSIF client might need to act asynchronously. WSIF provides two main features that meet this requirement:

- A **correlation service** that assigns identifiers to messages so that the request can match up with the (eventual) response.
- A **response handler** that picks up the response from the Web service at a later time.

For more information see the WSIF API topic [WSIFOperation - Asynchronous interactions](#) reference.

Developing the WSIF client - the Address Book Sample

The code fragments in this topic show you how to use the Web Services Invocation Framework (WSIF) API to invoke the `AddressBook` Sample Web service dynamically.

This is example code for dynamic invocation of the `AddressBook` sample Web service using WSIF:

```
try {
    String wsdlLocation="clients/addressbook/AddressBookSample.wsdl";

    // The starting point for any dynamic invocation using wsif is a
    // WSIFServiceFactory. We create ourselves one via the newInstance
```

```

// method.
WSIFServiceFactory factory = WSIFServiceFactory.newInstance();

// Once we have a factory, we can use it to create a WSIFService object
// corresponding to the AddressBookService service in the wsdl file.
// Note: since we only have one service defined in the wsdl file, we
// do not need to use the namespace and name of the service and can pass
// null instead. This also applies to the port type, although values have
// been used below for illustrative purposes.
WSIFService service = factory.getService(
    wsdlLocation, // location of the wsdl file
    null, // service namespace
    null, // service name
    "http://www.ibm.com/namespace/wsif/samples/ab", // port type namespace
    "AddressBookPT" // port type name
);

// The AddressBook.wsdl file contains the definitions for two complexType
// elements within the schema element. We will now map these complexTypes
// to Java classes. These mappings are used by the Apache SOAP provider
service.mapType(
    new javax.xml.namespace.QName(
        "http://www.ibm.com/namespace/wsif/samples/ab/types",
        "address"),
    Class.forName("com.ibm.www.namespace.wsif.samples.ab.types.WSIFAddress"));
service.mapType(
    new javax.xml.namespace.QName(
        "http://www.ibm.com/namespace/wsif/samples/ab/types",
        "phone"),
    Class.forName("com.ibm.www.namespace.wsif.samples.ab.types.WSIFPhone"));
// We now have a WSIFService object. The next step is to create a
// WSIFPort object for the port we wish to use. The
// getPort(String portName) method allows us to generate a WSIFPort
// from the port name.
WSIFPort port = null;

if (portName != null) {
    port = service.getPort(portName);
}
if (port == null) {
    // If no port name was specified, attempt to create a WSIFPort from
    // the available ports for the port type specified on the service
    port = getPortFromAvailablePortNames(service);
}

// Once we have a WSIFPort, we can create an operation. We are going to
// execute the addEntry operation and therefore we attempt to create a
// WSIFOperation corresponding to it. The addEntry operation is overloaded
// in the wsdl ie. there are two versions of it, each taking different
// parameters (parts). This overloading requires that we specify the input
// and output message names for the operation in the createOperation method
// so that the correct operation can be resolved. Since the addEntry
// operation has no output message, we use null for its name.
WSIFOperation operation =
    port.createOperation("addEntry", "AddEntryWholeNameRequest", null);

// Create messages to use in the execution of the operation. This should
// be done by invoking the createXXXXMessage methods on the WSIFOperation.
WSIFMessage inputMessage = operation.createInputMessage();
WSIFMessage outputMessage = operation.createOutputMessage();
WSIFMessage faultMessage = operation.createFaultMessage();

// Create a name and address to add to the addressbook
String nameToAdd="Chris P. Bacon";
WSIFAddress addressToAdd =
    new WSIFAddress (1,
        "The Waterfront",

```

```

        "Some City",
        "NY",
        47907,
        new WSIFPhone (765, "494", "4900"));
// Add the name and address to the input message
inputMessage.setObjectPart("name", nameToAdd);
inputMessage.setObjectPart("address", addressToAdd);

// Execute the operation, obtaining a flag to indicate its success
boolean operationSucceeded =
    operation.executeRequestResponseOperation(
        inputMessage,
        outputMessage,
        faultMessage);

if (operationSucceeded) {
    System.out.println(
        "Successfully added name and address to addressbook\n");
} else {
    System.out.println(
        "Failed to add name and address to addressbook");
}

// Start from fresh
operation = null;
inputMessage = null;
outputMessage = null;
faultMessage = null;

// This time we will lookup an address from the addressbook.
// The getAddressFromName operation is not overloaded in the
// wsdl and therefore we can simply specify the operation name
// without any input or output message names.
operation = port.createOperation("getAddressFromName");

// Create the messages
inputMessage = operation.createInputMessage();
outputMessage = operation.createOutputMessage();
faultMessage = operation.createFaultMessage();

// Set the name to find in the addressbook
String nameToLookup="Chris P. Bacon";
inputMessage.setObjectPart("name", nameToLookup);

// Execute the operation
operationSucceeded =
    operation.executeRequestResponseOperation(
        inputMessage,
        outputMessage,
        faultMessage);
if (operationSucceeded) {
    System.out.println(
        "Successful lookup of name '
"+nameToLookup+"' in addressbook"); // We can obtain the address that was
found by querying the output // message WSIFAddress addressFound =
(WSIFAddress) outputMessage.getObjectPart( "address"); System.out.println("The
address found was:"); System.out.println(addressFound); } else {
System.out.println("Failed to lookup name in addressbook"); } } catch (Exception e)
{ System.out.println("An exception occurred when running the sample:");
e.printStackTrace(); } }

```

The code above refers to the following Sample method:

```

WSIFPort getPortFromAvailablePortNames(WSIFService service)
    throws WSIFException {
    String portChosen = null;

    // Obtain a list of the available port names for the service
    Iterator it = service.getAvailablePortNames();
    {
        System.out.println("Available ports for the service are: ");
        while (it.hasNext()) {
            String nextPort = (String) it.next();
            if (portChosen == null)
                portChosen = nextPort;
            System.out.println(" - " + nextPort);
        }
    }
    if (portChosen == null) {
        throw new WSIFException("No ports found for the service!");
    }
    System.out.println("Using port " + portChosen + "\n");

    // An alternative way of specifying the port to use on the service
    // is to use the setPreferredPort method. Once a preferred port has
    // been set on the service, a WSIFPort can be obtained via getPort
    // (no arguments). If a preferred port has not been set and more than
    // one port is available for the port type specified in the WSIFService,
    // an exception is thrown.
    service.setPreferredPort(portChosen);
    WSIFPort port = service.getPort();
    return port;
}

```

The web service itself uses the following classes:

WSIFAddress:

```

public class WSIFAddress implements Serializable {

    //instance variables
    private int streetNum;
    private java.lang.String streetName;
    private java.lang.String city;
    private java.lang.String state;
    private int zip;
    private WSIFPhone phoneNumber;

    //constructors
    public WSIFAddress () { }

    public WSIFAddress (int streetNum,
                        java.lang.String streetName,
                        java.lang.String city,
                        java.lang.String state,
                        int zip,
                        WSIFPhone phoneNumber) {
        this.streetNum = streetNum;
        this.streetName = streetName;
        this.city = city;
        this.state = state;
        this.zip = zip;
        this.phoneNumber = phoneNumber;
    }

    public int getStreetNum() {
        return streetNum;
    }

    public void setStreetNum(int streetNum) {

```

```

        this.streetNum = streetNum;
    }

    public java.lang.String getStreetName() {
        return streetName;
    }

    public void setStreetName(java.lang.String streetName) {
        this.streetName = streetName;
    }

    public java.lang.String getCity() {
        return city;
    }

    public void setCity(java.lang.String city) {
        this.city = city;
    }

    public java.lang.String getState() {
        return state;
    }

    public void setState(java.lang.String state) {
        this.state = state;
    }

    public int getZip() {
        return zip;
    }

    public void setZip(int zip) {
        this.zip = zip;
    }

    public WSIFPhone getPhoneNumber() {
        return phoneNumber;
    }

    public void setPhoneNumber(WSIFPhone phoneNumber) {
        this.phoneNumber = phoneNumber;
    }
}

```

WSIFPhone:

```

public class WSIFPhone implements Serializable {

    //instance variables
    private int areaCode;
    private java.lang.String exchange;
    private java.lang.String number;

    //constructors
    public WSIFPhone () { }

    public WSIFPhone (int areaCode,
                      java.lang.String exchange,
                      java.lang.String number) {
        this.areaCode = areaCode;
        this.exchange = exchange;
        this.number = number;
    }

    public int getAreaCode() {
        return areaCode;
    }
}

```

```

    public void setAreaCode(int areaCode) {
        this.areaCode = areaCode;
    }

    public java.lang.String getExchange() {
        return exchange;
    }

    public void setExchange(java.lang.String exchange) {
        this.exchange = exchange;
    }

    public java.lang.String getNumber() {
        return number;
    }

    public void setNumber(java.lang.String number) {
        this.number = number;
    }
}

```

WSIFAddressBook:

```

public class WSIFAddressBook {
    private Hashtable name2AddressTable = new Hashtable();

    public WSIFAddressBook() {
    }

    public void addEntry(String name, WSIFAddress address)
    {
        name2AddressTable.put(name, address);
    }

    public void addEntry(String firstName, String lastName,
                          WSIFAddress address)
    {
        name2AddressTable.put(firstName+" "+lastName, address);
    }

    public WSIFAddress getAddressFromName(String name)
    throws IllegalArgumentException
    {
        if (name == null)
        {
            throw new IllegalArgumentException("The name argument must not be " +
                                             "null.");
        }
        return (WSIFAddress)name2AddressTable.get(name);
    }
}

```

And here's the corresponding WSDL file for the Web service:

```

<?xml version="1.0" ?>

<definitions targetNamespace="http://www.ibm.com/namespace/wsif/samples/ab"
    xmlns:tns="http://www.ibm.com/namespace/wsif/samples/ab"
    xmlns:typens="http://www.ibm.com/namespace/wsif/samples/ab/types"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:format="http://schemas.xmlsoap.org/wsdl/formatbinding/"
    xmlns:java="http://schemas.xmlsoap.org/wsdl/java/"
    xmlns:ejb="http://schemas.xmlsoap.org/wsdl/ejb/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">

```



```

<types>
  <xsd:schema
    targetNamespace="http://www.ibm.com/namespace/wsif/samples/ab/types"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <xsd:complexType name="phone">
      <xsd:element name="areaCode" type="xsd:int"/>
      <xsd:element name="exchange" type="xsd:string"/>
      <xsd:element name="number" type="xsd:string"/>
    </xsd:complexType>

    <xsd:complexType name="address">
      <xsd:element name="streetNum" type="xsd:int"/>
      <xsd:element name="streetName" type="xsd:string"/>
      <xsd:element name="city" type="xsd:string"/>
      <xsd:element name="state" type="xsd:string"/>
      <xsd:element name="zip" type="xsd:int"/>
      <xsd:element name="phoneNumber" type="typens:phone"/>
    </xsd:complexType>

  </xsd:schema>
</types>

<message name="AddEntryWholeNameRequestMessage">
  <part name="name" type="xsd:string"/>
  <part name="address" type="typens:address"/>
</message>

<message name="AddEntryFirstAndLastNamesRequestMessage">
  <part name="firstName" type="xsd:string"/>
  <part name="lastName" type="xsd:string"/>
  <part name="address" type="typens:address"/>
</message>

<message name="GetAddressFromNameRequestMessage">
  <part name="name" type="xsd:string"/>
</message>

<message name="GetAddressFromNameResponseMessage">
  <part name="address" type="typens:address"/>
</message>

<portType name="AddressBookPT">
  <operation name="addEntry">
    <input name="AddEntryWholeNameRequest"
      message="tns:AddEntryWholeNameRequestMessage"/>
  </operation>
  <operation name="addEntry">
    <input name="AddEntryFirstAndLastNamesRequest"
      message="tns:AddEntryFirstAndLastNamesRequestMessage"/>
  </operation>
  <operation name="getAddressFromName">
    <input name="GetAddressFromNameRequest"
      message="tns:GetAddressFromNameRequestMessage"/>
    <output name="GetAddressFromNameResponse"
      message="tns:GetAddressFromNameResponseMessage"/>
  </operation>
</portType>

<binding name="SOAPHttpBinding" type="tns:AddressBookPT">
  <soap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="addEntry">
    <soap:operation soapAction=""/>
    <input name="AddEntryWholeNameRequest">
      <soap:body use="encoded"

```

```

        namespace="http://www.ibm.com/namespace/wsif/samples/ab"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    </input>
</operation>
<operation name="addEntry">
    <soap:operation soapAction=""/>
    <input name="AddEntryFirstAndLastNamesRequest">
        <soap:body use="encoded"
            namespace="http://www.ibm.com/namespace/wsif/samples/ab"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    </input>
</operation>
<operation name="getAddressFromName">
    <soap:operation soapAction=""/>
    <input name="GetAddressFromNameRequest">
        <soap:body use="encoded"
            namespace="http://www.ibm.com/namespace/wsif/samples/ab"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    </input>
    <output name="GetAddressFromNameResponse">
        <soap:body use="encoded"
            namespace="http://www.ibm.com/namespace/wsif/samples/ab"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    </output>
</operation>
</binding>

<binding name="JavaBinding" type="tns:AddressBookPT">
    <java:binding/>
    <format:typeMapping encoding="Java" style="Java">
        <format:typeMap typeName="typens:address"
            formatType="com.ibm.www.namespace.wsif.samples.ab.types.WSIFAddress"/>
        <format:typeMap typeName="xsd:string" formatType="java.lang.String"/>
    </format:typeMapping>
    <operation name="addEntry">
        <java:operation
            methodName="addEntry"
            parameterOrder="name address"
            methodType="instance"/>
        <input name="AddEntryWholeNameRequest"/>
    </operation>
    <operation name="addEntry">
        <java:operation
            methodName="addEntry"
            parameterOrder="firstName lastName address"
            methodType="instance"/>
        <input name="AddEntryFirstAndLastNamesRequest"/>
    </operation>
    <operation name="getAddressFromName">
        <java:operation
            methodName="getAddressFromName"
            parameterOrder="name"
            methodType="instance"
            returnPart="address"/>
        <input name="GetAddressFromNameRequest"/>
        <output name="GetAddressFromNameResponse"/>
    </operation>
</binding>

<binding name="EJBBinding" type="tns:AddressBookPT">
    <ejb:binding/>
    <format:typeMapping encoding="Java" style="Java">
        <format:typeMap typeName="typens:address"
            formatType="com.ibm.www.namespace.wsif.samples.ab.types.WSIFAddress"/>
        <format:typeMap typeName="xsd:string" formatType="java.lang.String"/>
    </format:typeMapping>
    <operation name="addEntry">

```

```

    <ejb:operation
      methodName="addEntry"
      parameterOrder="name address"
      interface="remote"/>
    <input name="AddEntryWholeNameRequest"/>
  </operation>
  <operation name="addEntry">
    <ejb:operation
      methodName="addEntry"
      parameterOrder="firstName lastName address"
      interface="remote"/>
    <input name="AddEntryFirstAndLastNamesRequest"/>
  </operation>
  <operation name="getAddressFromName">
    <ejb:operation
      methodName="getAddressFromName"
      parameterOrder="name"
      interface="remote"
      returnPart="address"/>
    <input name="GetAddressFromNameRequest"/>
    <output name="GetAddressFromNameResponse"/>
  </operation>
</binding>
<service name="AddressBookService">
  <port name="SOAPPort" binding="tns:SOAPHttpBinding">
    <soap:address
      location="http://localhost/wsif/samples/addressbook/soap/servlet/rpcrouter"/>
  </port>
  <port name="JavaPort" binding="tns:JavaBinding">
    <java:address className="services.addressbook.WSIFAddressBook"/>
  </port>
  <port name="EJBPort" binding="tns:EJBBinding">
    <ejb:address className="services.addressbook.ejb.AddressBookHome"
      jndiName="ejb/samples/wsif/AddressBook"
      classLoader="services.addressbook.ejb.AddressBook.ClassLoader"/>
  </port>
</service>
</definitions>

```

Using complex types

WSIF supports the use of user-defined complex types through the mapping of complex types to Java classes.

This mapping can be specified manually or automatically as described in the sections below:

- Manual mapping of complex types.

Note: Any calls to the WSIFService `mapType` and `mapPackage` methods used for manual mapping will override any equivalent mapping information that is produced automatically. This helps to maintain backwards compatibility, and also allows for less standard mappings.

Manual mapping of complex types

The method to use to create these mappings by hand depends on the provider being used. For the Java and EJB providers, the mappings are specified in the `wsdl` file in the binding element. The syntax for specifying the mapping is as follows:

```

<binding .... >
  <ejb:binding|java:binding/>
    <format:typeMapping style="Java" encoding="Java"/>?

```

```

        <format:typeMap name="qname" formatType="nmtoken"/>*
    </format:typeMapping>
    ...
</binding>

```

where “?” means “optional” and “*” means “0 or more”.

The `format:typeMap` **name** attribute is a qualified name of a complex type or simple type used by one of the operations.

The `format:typeMap` **formatType** attribute is the fully qualified class name for the Java class that the element specified by **name** maps to.

If using the Apache SOAP provider then the mapping of a complex type to a Java Class is specified in the client code through two methods on the `org.apache.wsif.WSIFService` interface:

```
public void mapType(QName elementType, Class javaType)
```

and

```
public void mapPackage(String namespaceURI, String packageName)
```

The **mapType** allows the user to specify a mapping between a WSDL element and method takes a `QName` representing the complex type or simple type and the corresponding Java class to which it maps.

The **mapPackage** method allows the user to specify a more general mapping between a namespace and a Java package. Any custom complex or simple types whose namespace matches that of the mapping will be mapped to a Java class in the corresponding package. The name of the actual class is derived from the name of the complex type using standard xml to Java naming conventions.

Using the Java Naming and Directory Interface (JNDI)

This example task shows you how to use WSIF to bind a reference to a Web service, then look up the reference using JNDI.

You access a Web service through information given in the WSDL document for the service. If you don't know where to find the WSDL document for the service, but you know that it has been registered in a UDDI registry, then you look it up in the registry. Java programs access java objects and resources in a similar manner, but using a JNDI interface.

The following example shows how, using the Web Services Invocation Framework (WSIF), you can bind a reference to a Web service then look up the reference using JNDI.

Specifying the argument values for the Web service

The Web service is represented in WSIF by an instance of the `org.apache.wsif.naming.WSIFServiceRef` class. This simple Referenceable object has the following constructor:

```
public WSIFServiceRef(
    String WSDL,
    String SNS,
    String sName,
    String ptNS,
```

```

        String ptName)
    {
        [...]
    }

```

where

- *WSDL* is the location of the WSDL file containing the definition of the service.
- *sNS* is the full namespace for the service definition (null can be specified if only one service is defined in the WSDL file).
- *sName* is the local name for the service definition (null can be specified if only one service is defined in the WSDL file).
- *ptNS* is the full namespace for the port type within the service that you want to use (null can be specified if only one port type is available for the service).
- *ptName* is the local name for the port type (null can be specified if only one port type is available for the service).

For example, if the WSDL file for the Web service is available from the Web address `http://localhost/WSDL/Example.WSDL` and contains the following service and port type definitions -

```

<definitions targetNamespace="http://hostname/namespace/example"
    xmlns:abc="http://hostname/namespace/abc"
[...]
    <portType name="ExamplePT">
        <operation name="exampleOp">
            <input name="exampleInput" message="tns:ExampleInputMsg"/>
        </operation>
    </portType>
[...]
    <service name="abc:ExampleService">
[...]
    </service>
[...]
</definitions>

```

- then you specify the following argument values for `WSIFServiceRef`:

- *WSDL* is `http://localhost/WSDL/Example.WSDL`
- *sNS* is `http://hostname/namespace/abc`
- *sName* is `ExampleService`
- *ptNS* is `http://hostname/namespace/example`
- *ptName* is `ExamplePT`

Binding the service using JNDI

To bind the service reference in the naming directory using JNDI, you can use the WebSphere Application Server `JndiHelper` `com.ibm.websphere.naming.JndiHelper` class as follows:

```

[...]
import com.ibm.websphere.naming.JndiHelper;
import org.apache.wsif.naming.*;
[...]
try {
    Context startingContext = new InitialContext();
    WSIFServiceRef ref = new WSIFServiceRef("http://localhost/WSDL/Example.WSDL",
        "http://hostname/namespace/abc",
        "ExampleService",
        "http://hostname/namespace/example",
        "ExamplePT");
    JndiHelper.recursiveRebind(startingContext,
        "myContext/mySubContext/myServiceRef", ref);
}

```

```

        catch (NamingException e) {
            // Handle error.
        }
    [...]

```

Looking up the service using JNDI

The following code fragment shows the lookup of a service using JNDI:

```

[...]
    try {
[...]
        InitialContext ic = new InitialContext();
        WSIFService myService =
            (WSIFService) ic.lookup("myContext/mySubContext/myServiceRef");
[...]
    }
    catch (NamingException e) {
        // Handle error.
    }
[...]
```

Interacting with the J2EE container in WebSphere Application Server

Interaction with a container is limited to the following aspects:

1. The WebSphere Application Server administrative console and WCCM allow users to define Web services to WebSphere. This is described in Using the Java Naming and Directory Interface (JNDI) and WSIF system management and administration. As part of the definition of a service, the administrator may define a “preferred port”.
2. The Web Services Invocation Framework (WSIF) makes log and trace calls to the WebSphere Server JRAS services, as described in Trace and logging for WSIF.
3. Some providers use the Java 2 platform, Enterprise Edition (J2EE) programming model to utilize J2EE services. The EJB provider uses the Java Naming and Directory Interface (JNDI) and calls to remote EJBs.
4. WSIF wraps the use of container services so that when WSIF is run in an unmanaged (thin) environment, the operation can succeed.

Running WSIF as a client

WSIF runs in the WebSphere Application Server application client container, and in similar clients from other suppliers.

To simplify the process of launching client applications in the WebSphere Application Server application client, use the `launchClient` tool as described in Running application clients.

WSIF system management and administration

The Web Services Invocation Framework (WSIF) is provided as a stand-alone JAR file called `wsif.jar`. The JAR file contains the core WSIF classes, and the Java, EJB, SOAP over HTTP and SOAP over JMS providers. Additional providers are packaged as separate JAR files.

When you install WebSphere Application Server, `wsif.jar` is put on the WebSphere or Java Virtual Machine (JVM) class path.

WSIF requires no further configuration. WSIF is a thin abstraction layer between application code and the relevant invocation infrastructure.

For specific information on other aspects of managing your WSIF system, see the following topics:

- Maintaining the WSIF properties file.
- Enabling security for WSIF.
- Trace and logging for WSIF.
- WSIF troubleshooting tips.
- WSIF (Web Services Invocation Framework) messages.

Maintaining the WSIF properties file

The Web Services Invocation Framework (WSIF) properties are stored in a properties file (in `wsif.jar`) called `wsif.properties`. This file is kept on the class path, so that WSIF can find it and the client administrator can use it to configure WSIF.

The initial contents of `wsif.properties` are printed out below. All the possible properties are listed and described.

```
# Two properties are used to override which WSIFProvider is selected when there
# exists multiple providers supporting the same namespace URI. These
# properties are:
#
#   wsif.provider.default.CLASSNAME=N
#   wsif.provider.uri.M.CLASSNAME=URI
#
# CLASSNAME is the WSIFProvider class name
# N is the number of following default wsif.provider.uri.M.CLASSNAME properties
# M is a number from 1 to N to uniquely identify each
# wsif.provider.uri.M.CLASSNAME property key.
# For example the following two properties would override the default SOAP
# provider to be the Apache SOAP provider:
#
# wsif.provider.default.org.apache.wsif.providers.soap.ApacheSOAP
#   .WSIFDynamicProvider_ApacheSOAP=1
# wsif.provider.uri.1.org.apache.wsif.providers.soap.ApacheSOAP
#   .WSIFDynamicProvider_ApacheSOAP=\
# http://schemas.xmlsoap.org/wsdl/soap/
#
#
# maximum number of milliseconds to wait for a response to a synchronous
# request. Default value if not defined is to wait forever.
wsif.syncrequest.timeout=10000
#
# maximum number of seconds to wait for a response to an async request.
# if not defined on invalid defaults to no timeout
wsif.asyncrequest.timeout=60
```

Enabling security for WSIF

The Web Services Invocation Framework (WSIF) interacts with a security manager as follows:

- WSIF runs in the current Java 2 platform, Enterprise Edition (J2EE) security context without modifying it.
- When WSIF is run under a J2EE container, port implementations can utilize security context to pass on security tokens or credentials as necessary.

- WSIF implementations can automatically convert J2EE security context into appropriate context for onward services.

For WSIF to interact effectively with the WebSphere Application Server's security manager, the following permissions must be set in the server.policy file:

- FilePermission to load the WSDL (this is only required when a WSDL file is referred to using the file:/// protocol)
- RuntimePermission "getClassLoader" for the current thread's context class loader (this is not required by the EJB portion).
- RuntimePermission "accessDeclaredMembers" (this is required by both portions for handling enterprise beans)
- PropertyPermission for system properties (this is required by SOAP and many others; read and write access is required for the SOAP and Java portion, only write access is required for the EJB portion)
- NetPermission "specifyStreamHandler" (this must be in either the SOAP and Java portion, or the EJB portion, but it need not be in both).
- SocketPermission "host_name", "resolve" (this is not required by the SOAP and Java portion)
- SocketPermission "host_name:port_no", "connect" (this is required by both portions)

where *host_name* is your host name (for example localhost), and *port_no* is your port number (for example 9080).

WSIF troubleshooting tips

For information on resolving WebSphere-level problems, see Diagnosing and fixing problems.

To identify and resolve Web Services Invocation Framework (WSIF)-related problems, you can use the standard WebSphere Application Server trace and logging facilities. If you encounter a problem that you think might be related to WSIF, you can check for error messages in the WebSphere Application Server administrative console, and in the application server stdout.log file. You can also enable the application server debug trace to provide a detailed exception dump.

A list of the WSIF run-time system messages, with details of what each message means, is given in Message reference for WSIF.

Here is a checklist of major WSIF activities, with advice on common problems associated with each activity:

Create service

Handcrafted WSDL can cause numerous problems. To help ensure that your WSDL is valid, use a tool such as WebSphere Studio to create your service.

Define transport mechanism

For the Java Messaging Service (JMS), check that you have set up the Java Naming and Directory Interface (JNDI) correctly, and created all the necessary queues.

For SOAP, make sure that the deployment descriptor (dds.xml) is correct - preferably by creating it using WebSphere Studio or similar tooling.

Create client - Java code

Follow the correct format for creating a WSIF service, port, operation and message. For examples of correct code, see the Address Book Sample.

Compile code (client and service)

Check that the build path against code is correct, and that it contains the correct levels of JAR files.

Create a valid EAR file for your service in preparation for deployment to a Web server.

Deploy service

When you install and deploy the service EAR file, check carefully any messages given when the service is deployed.

Server setup and start

Make sure that the WebSphere Application Server file `server.policy` (in the `/properties` directory) has the correct security settings within it. For more information see *Enabling security for WSIF*.

WSIF setup

Check that file `wsif.properties` is correctly set up. For more information see *Maintaining the WSIF properties file*.

Run client

Either check that you have defined the class path correctly to include references to your client classes, WSIF JAR files and any other necessary JAR files, or (preferably) run your client using the WebSphere Application Server `launchClient` tool.

Here is a list of common errors, and information on their probable causes:

- **“No class definition” errors received when running client code.**

This is likely to be a problem with the class path setup. Check that the relevant JAR files are included.

- **“Cannot find WSDL” error.**

Some likely causes are:

- The application server is not running.
- The server location and port number in the WSDL are not correct.
- The WSDL is badly formed (check the error messages in the application server `stdout.log` file).
- The application server has not been restarted since the service was installed.

You might also try the following checks:

- Can you load the WSDL into your Web browser from the location specified in the error message?
- Can you load the corresponding WSDL binding files into your Web browser?
- **Your Web service EAR file does not install correctly onto the application server.**

It is likely that the EAR file is badly formed. Verify the installation by completing the following steps:

- For an EJB binding, run the WebSphere Application Server tool `\bin\dumpnamespace`. This tool lists the current contents of the JNDI directory.
- For a SOAP over HTTP binding, open `http://pathToServer/WebServiceName/admin/list.jsp` (if you have the SOAP administration pages installed). This page lists all currently installed Web services.
- For a SOAP over JMS binding, complete the following checks:
 - Check that the Queue Manager is running.
 - Check that the necessary queues are defined.
 - Check the JNDI setup.
 - Use the `jmsadmin` tool “display context” option to list the current JNDI definitions.
 - Check that the Remote Procedure Call (RPC) router is running.

- **There is a permissions problem or security error.**

Check that the WebSphere Application Server file `server.policy` (in the `/properties` directory) has the correct security settings within it. For more information see [Enabling security for WSIF](#).

- **Using WSIF with multiple clients causes a SOAP parsing error.**

Before you deploy a Web service to WebSphere Application Server, you must decide on the scope of the Web service. The deployment descriptor file (`dds.xml`) for the Web service includes the following line:

```
<isd:provider type="java" scope="Application" .....
```

You can set Scope to “Application” or “Session”. The default setting is “Application”, and this is correct if each request to the Web service does not require objects to be maintained for longer than a single instance. If Scope is set to “Application” the objects are not available to another request during the execution of the single instance, and they are released on completion. If your Web service needs objects to be maintained for multiple requests, and to be unique within each request, you must set the scope to “Session”. If Scope is set to “Session”, the objects are not available to another request during the life of the session, and they are released on completion of the session. If scope is set to “Application” but it should be set to “Session”, you might get the following SOAP error:

```
SOAPException: SOAP-ENV:ClientParsing error, response was:
FWK005 parse may not be called while parsing.;
nested exception is:
```

```
[SOAPException: faultCode=SOAP-ENV:Client; msg=Parsing error, response was:
```

```
FWK005 parse may not be called while parsing.;
targetException=org.xml.sax.SAXException:
FWK005 parse may not be called while parsing.]
```

Trace and logging for WSIF

If you want to enable trace for the Web Services Invocation Framework (WSIF) API within WebSphere Application Server, and have trace, stdout and stderr for the application server written to a well-known location, see [Enabling traceSetting up component trace \(CTRACE\)](#).

WSIF offers tracepoints at opening and closing of ports, invocation of services, and responses from services.

To trace the WSIF API, you need to specify the following trace string:

```
wsif=all=enabled
```

WSIF also includes a SimpleLog utility through which you can run trace when using WSIF outside of WebSphere Application Server. To enable this, complete the following steps:

1. create file `commons-logging.properties` with the following contents (shown here on multiple lines for publication):

```
org.apache.commons.logging.LogFactory=
  org.apache.commons.logging.impl.LogFactoryImpl
org.apache.commons.logging.Log=org.apache.commons.logging.impl.SimpleLog
```

2. create file `simplelog.properties` with the following contents:

```
org.apache.commons.logging.simplelog.defaultlog=trace
org.apache.commons.logging.simplelog.showShortLogname=true
org.apache.commons.logging.simplelog.showdatetime=true
```

3. Put both these files, and commons-logging.jar, on the class path.

The SimpleLog mechanism writes trace to file System.err.

WSIF (Web Services Invocation Framework) messages

This topic contains a list of the WSIF run-time system messages, with details of what each message means.

WebSphere system messages are logged from a variety of sources, including application server components and applications. 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.

For more information about the message identifier format, see the topic Message reference.

WSIF0001E: An extension registry was not found for the element type "{0}"

Explanation: Parameters: {0} element type. No extension registry was found for the element type specified.

User Response: Add the appropriate extension registry to the port factory in your code.

WSIF0002E: A failure occurred in loading WSDL from "{0}"

Explanation: Parameters: {0} location of the WSDL file. The WSDL file could not be found at the location specified or did not parse correctly

User Response: Check that the location of the WSDL file is correct. Check that any network connections required are available. Check that the WSDL file contains valid WSDL.

WSIF0003W: An error occurred finding pluggable providers: {0}

Explanation: Parameters: {0} specific details about the error. There was a problem locating a WSIF pluggable provider using the J2SE 1.3 JAR file extensions to support service providers architecture. The WSIF trace file will contain the full exception details.

User Response: Verify that a META-INF/services/org.apache.wsif.spi.WSIFProvider file exists in a provider jar, that each class referenced in the META-INF file exists in the class path, and that each class implements org.apache.wsif.spi.WSIFProvider. The class in error will be ignored and WSIF will continue locating other pluggable providers.

WSIF0004E: WSDL contains an operation type "{0}" which is not supported for "{1}"

Explanation: Parameters: {0} name of the operation type specified. {1} name of the portType for the operation. An operation type which is not supported has been specified in the WSDL.

User Response: Remove any operations of the unsupported type from the WSDL. If the operation is required then make sure all messages have been correctly specified for the operation.

WSIF0005E: An error occurred when invoking the method "{1}" . (" {0} ")

Explanation: Parameters: {0} name of communication type. For example EJB or Apache SOAP. {1} name of the method that failed. An error was encountered when invoking a method on the web service using the communication shown in brackets.

User Response: Check that the method exists on the web service and that the correct parts have been added to the operation as described in the WSDL. Network problems might be a cause if the method is remote and so check any required connections.

WSIF0006W: Multiple WSIFProvider found supporting the same namespace URI “{0}” . Found (“{1}”)

Explanation: Parameters: {0} the namespace URI. {1} a list of the WSIFProvider found.. There are multiple org.apache.wsif.spi.WSIFProvider classes in the service provider path that support the same namespace URI.

User Response: A following WSIF0007I message will be issued notifying which WSIFProvider will be used. Which WSIFProvider is chosen is based on settings in the wsif.properties file, or if not defined in the properties, the last WSIFProvider found will be used. See the wsif.properties file for more details on how to define which provider should be used to support a namespace URI.

WSIF0007I: Using WSIFProvider “{0}” for namespaceURI “{1}”

Explanation: Parameters: {0} the classname of the WSIFProvider being used. {1} the namespaceURI the provider will be used to support.. Either a previous WSIF0006W message has been issued or the SetDynamicWSIFProvider method has been used to override the provider used to support a namespaceURI.

User Response: None. See also WSIF0006W.

WSIF0008W: WSIFDefaultCorrelationService removing correlator due to timeout. ID:“{0}”

Explanation: Parameters: {0} the id of the correlator being removed from the correlation service. A stored correlator is being removed from the correlation service due to its timeout expiring.

User Response: Determine why no response has been received for the asynchronous request within the timeout period. The wsif.asyncrequest.timeout property of the wsif.properties file defines the length of the timeout period.

WSIF0009I: Using correlation service - “{0}”

Explanation: Parameters: {0} the name of the correlation service being used. This identifies the name of the correlation service that will be used to process asynchronous requests.

User Response: None. If a correlation service other than the default WSIF supplied one is required, ensure that it is correctly registered in the JNDI java:comp/wsif/WSIFCorrelationService namespace.

WSIF0010E: Exception thrown while processing asynchronous response - “{0}”

Explanation: Parameters: {0} the error message string of the exception. While processing the response from an executeRequestResponseAsync call an exception was thrown.

User Response: Use the exception error message string to determine the cause of the error. The WSIF trace will have more details on the error including the exception stack trace.

WSIF0011I: Preferred port “{0}” was not available

Explanation: Parameters: {0} the user’s preferred port. The preferred port set by the user on org.apache.wsif.WSIFService is not available

User Response: None unless this message appears for long periods of time in which case the user might want to pick a different port as their preferred port.

WSIF API

The Web Services Invocation Framework (WSIF) API supports the invocation of Services defined in WSDL. WSIF is intended to be used in both WSIF clients and also in Web service intermediaries.

The WSIF API is driven by the abstract service description in WSDL; it is completely independent of the actual binding used. This makes the API more natural to work with, because it uses WSDL terms to refer to message parts, operations, and so on.

The WSIF API was designed for the WSDL usage model; to pick a port that supports the port type needed, then invoke the operation by providing the necessary abstract input message consisting of the required parts, without worrying about how the message is mapped to a specific binding protocol.

Other Web service APIs, for example SOAP APIs, are not designed on WSDL, but for a specific binding protocol with its associated syntax; for example, target URIs and encoding styles.

The WSIF API's main interfaces are described in the following topics:

- Creating a message for sending to a port (the `WSIFMessage` interface).
- WSIF API reference: Finding a port factory or service (the `WSIFService` interface and the `WSIFServiceFactory` class).
- WSIF API reference: Using ports (the `WSIFPort` interface and the `WSIFOperation` interface).

Note: You must ensure that your application uses only one thread to call WSIF.

For additional technical details of the WSIF API, see the WSIF Javadoc.

WSIF API reference: Creating a message for sending to a port

For message management (that is, message construction and parsing) the underlying API is modeled on WSDL semantics. There is a simple and direct mapping from the WSDL model to the Web Services Invocation Framework (WSIF) classes.

In WSDL, a `Message` describes the abstract type of the input or output to an operation. The corresponding WSIF class is **`WSIFMessage`**, which represents in memory the actual input or output of an operation. A `WSIFMessage` is a container for a set of named parts. The `WSIFMessage` interface separates the actual representation of the data from the abstract type defined by WSDL. WSDL defines Messages as XML Schema types. There are two natural ways to represent a WSDL message in a run-time environment:

- The generated Java class, based on a WSDL to Java mapping such as that provided by a Java API for XML-based remote procedure call (JAX-RPC).
- The XML representation of the data, for example using SOAP Encoding.

Each option offers benefits in different scenarios. The Java class is the natural approach when WSIF is being used in a standard Java client. However, in other scenarios where WSIF is being used in an intermediary, it may be more efficient to keep a WSDL Message in the SOAP encoded format.

The style used to define messages must be consistent within the message, so all the parts in one message must be consistent. A string - `getRepresentationStyle()` - always returns `null`. This indicates that parts on this `WSIFMessage` are represented as Java objects.

Parts are added to a `WSIFMessage` with **`setObjectPart`** or **`setTypePart`**. Each part is named. Part names within a message are unique. If a part is **`set`** multiple times, the last time is the one that stands.

Parts are retrieved from a `WSIFMessage` by name with **`getObjectPart`** or **`getTypePart`**. If the named part does not exist, the `getXXXXPart` method returns a `WSIFException`.

Parts can be retrieved from the `Message` by use of Iterators through the **`getParts()`** and **`getPartNames()`** methods.

The order in which parts are set is not currently important, but the `Message` implementation may be more efficient if the parts are set in the parameter order specified by WSDL.

`WSIFMessages` are cloneable and serializable. If the parts set are not cloneable, the implementation should try to clone them using serialization. If the parts are not serializable either, then a `CloneNotSupportedException` will be thrown if cloning is attempted.

`WSIFMessages` can be sent between Java Virtual Machines (JVMs).

In addition to the containing parts, a `WSIFMessage` also has a message name. This is required for operation overloading - which is supported by WSDL and WSIF.

Here is the Javadoc for the `WSIFMessage` interface.

WSIF API reference: Finding a port factory or service

To find a port you use `WSIFService`, which is a factory for ports.

The port factory models and supports the WSDL approach in which a service is available on one or more ports. The factory hides the implementation of the port from the user. The Web Services Invocation Framework (WSIF) supports “dynamic” ports that are based on a particular protocol/transport and configured using the WSDL at run-time. For example, the dynamic SOAP port can invoke any SOAP service based on the WSDL description of that service. The Service allows you to hide and modify this at run-time.

Here is the `WSIFService` interface.

To find a service from a WSDL document at a Web address, or from a code-generated codebase, you can use the `WSIFServiceFactory` class.

WSIFService interface

The `WSIFService` is responsible for generating an instance of `WSIFOperation` to be used for a particular invocation of a service operation.

The Web Services Invocation Framework (WSIF) service stores a list of providers that can each generate a WSIF operation for a particular WSDL binding. This service looks up providers by the provider type, so, for example, it knows about one provider that handles SOAP ports, and other providers that handle Java ports that you define. In a managed environment, the container can configure `WSIFService`.

Here is the Javadoc for the `WSIFService` interface.

A `WSIFService` implementation can choose a preferred port based on a number of criteria. The `WSIFService` implementation can set the preferred port, or it can be set by calling **`setPreferredPort`**.

The **`getPort`** method returns an instance of `WSIFPort` that is used to invoke a service on the port. Variants of the **`getPort`** method are used to define the

characteristics of the port to be created. **getPort** with no arguments returns the “preferred” port. **getPort** with a string argument returns the port named by the string containing the WSDL identifier for the selected port. The return value is null if the port name is not valid.

If a port is chosen (either by WSIFService default behavior, or by setPreferredPort), then the WSIFService implementation validates that the relevant provider exists and is configured. If the provider fails this validation check, WSIFService chooses any other port whose provider is defined. For example, if the preferred port is SOAP over JMS, but the JMS libraries are not available, WSIF chooses another port. If no preferred port is set, or the preferred port is not available, the WSIF implementation chooses the first available port listed in the WSDL.

The method **getAvailablePortNames()** returns an Iterator of Strings which is the list names of WSDL ports, filtered by the set of available providers.

The **getDefinition()** method returns the WSDL definition that was used to provide the Service. If the WSDL definition is not available, this returns a null.

WSIFServiceFactory class

To find a service from a WSDL document at a Web address, or from a code-generated codebase, you can use the WSIFServiceFactory class.

Note: When creating a WSIFService from a WSIFServiceFactory you can specify a ClassLoader object to use in locating the WSDL file. You need to do this when the WSDL file is in a JAR file. In such a case, the location of the WSDL file is specified relative to the root of the JAR file, using forward slashes with the preceding slash removed.

For example:

```
com/myCompany/wsd1/MyWSDLFile.wsdl
```

rather than

```
/com/myCompany/wsd1/MyWSDLFile.wsdl
```

Here is the Javadoc for the WSIFServiceFactory class.

The WSIFServiceFactory returns a null if no service is found with that identifier.

WSIF API reference: Using ports

A WSIFPort handles the details of invoking an operation. The port provides access to the actual implementation of the service.

A WSDL document can provide many different WSDL bindings, and these bindings can drive multiple ports. The client can choose a port, the service stub can choose a port, or the Web Services Invocation Framework (WSIF) can choose a default port.

The port offers an interface to retrieve an Operation object. A WSIFOperation offers the ability to execute the given operation.

Note:

If the port is serialized and deserialized at a later time, then WSIF ensures that the client provides the correct information to the server to identify the instance. If the server instance is no longer available, then it is up to the

server to decide whether to throw a fault or provide a new instance. That behavior can depend on the type of service.

For example, for an enterprise bean the WSIFPort stores the EJB Home, and uses it to “select” the bean before each invocation. It is the responsibility of the client to serialize or maintain the port instance if it wants instance support. The client must create a new operation and messages for each invocation.

Here is the WSIFPort interface.

Here is the WSIFOperation interface.

WSIFPort interface

The port implements a factory method for WSIFOperations.

Here is the Javadoc for the WSIFPort interface.

The `createOperation(String)` method returns a new instance of a WSIFOperation object. If the `operationName` is not valid, it will throw an exception. If the operation is overloaded, it will throw an exception. The `createOperation(String, String, String)` allows the Web Services Invocation Framework (WSIF) to support overloaded WSDL operations.

Note: You can overload based on the input parameters, but not on the output parameters.

It is the duty of the client to call the **close** method when a port is no longer in use. In many cases, where the transport is sessionless, like HTTP, this will have no effect. However, if the port is using a session-based protocol such as MQSeries, Java Messaging Service (JMS), or External Call Interface (ECI), this allows the port to cache an open connection to the server and then close it as required. Responsibly-written applications will call the close method if appropriate.

WSIFOperation interface

WSIFOperation is responsible for invoking a service based on a particular binding. WSIFOperation is the run-time representation of an operation. It provides methods to create input, output and fault messages, and to invoke the operation.

Here is the Javadoc for the WSIFOperation interface.

The **createInputMessage**, **createOutputMessage**, and **createFaultMessage** methods are factory methods to create the messages required by the invocation methods. All invocation methods require an input message.

The **executeRequestResponseOperation** invokes “In Out” operations.

The **executeInputOnlyOperation** invokes “In only” operations.

If the invocation method is **executeRequestResponseOperation** then an output and a fault message are instantiated and passed on the call to the **executeRequestResponseOperation** method. The output message contains the response message when the **executeRequestResponseOperation** returns true. If **executeRequestResponseOperation** returns false, then a fault occurred and is returned in the fault message.

All of these **executeNmmn** methods fail with an exception if there is an error in processing the request in the WSIF provider.

The **executeRequestResponseAsync** is a feature that allows “In Out” operations to be invoked with the reply handled using an alternate thread.

Use of the **setContext** and **getContext** methods is discussed in WSIFOperation - Context.

Use of the **executeRequestResponseAsync** feature is discussed further in WSIFOperation - Asynchronous interactions.

Setting the timeouts for synchronous and asynchronous operations is discussed in WSIFOperation - Synchronous and asynchronous timeouts.

WSIFOperation - Context: Although WSDL does not define context, a number of uses of the Web Services Invocation Framework (WSIF) require the ability to pass context to the port that is invoking the service. For example, a SOAP over HTTP port may require an HTTP username and password. This information is specific to the invocation, but not a parameter of the service. In general, context is defined as a set of name-value pairs. However, because Web services tend to define the types of data using XML Schema types, WSIF represents the name-value pairs of the context using the same representation that WSIFMessages use; that is a set of named Parts, each of which equates to an instance of an XML Schema type.

The WSIFOperation methods **setContext** and **getContext** allow you to pass context information to the binding. The port implementation can use this context - for example to update a SOAP header. There is no definition of how a port can utilize the context.

The parameter of the **setContext** and **getContext** methods is a WSIFMessage, and this has named parts defining the context information. The WSIFConstants class defines constants for the part names that can be set in a context WSIFMessage.

The following code shows how to set the user name and password to be used for HTTP basic authentication:

```
// set a basic authentication header
WSIFMessage headers = new WSIFDefaultMessage();
headers.setObjectPart( WSIFConstants.CONTEXT_HTTP_USER, "username" );
headers.setObjectPart( WSIFConstants.CONTEXT_HTTP_PSWD, "password" );
operation.setContext( headers );
```

The WSIFOperation will ignore context parts that it does not support, for example, the above code would be ignored by the WSIF Java provider.

The WSIFConstants class includes the following constants which can be used for context part names:

- CONTEXT_HTTP_USER
- CONTEXT_HTTP_PSWD
- CONTEXT_SOAP_HEADERS

The HTTP header values are expected to have a type **String**, and the SOAP header value is expected to have a type of **java.util.List** which should contain entries of type **org.w3c.dom.Element**.

WSIFOperation - Asynchronous interactions reference: The Web Services Invocation Framework (WSIF) allows asynchronous operation. In this mode of

operation, the client puts the request message as part of one transaction, and carries on with the thread of execution. The response message is then handled by a different thread, with a separate transaction. The SOAP over JMS and native JMS providers are the only WSIF providers that currently support asynchronous operation.

The WSIFPort has method **supportsAsync** to test if asynchronous operation is supported.

An asynchronous operation is initiated with the WSIFOperation method **executeRequestResponseAsync**. This method is designed to let a Remote Procedure Call (RPC) method be invoked asynchronously. The method returns before the operation has completed, and the thread of execution continues.

The response to the asynchronous request is processed by one of the WSIFOperation methods **fireAsyncResponse** or **processAsyncResponse**.

To initiate the request, there are two forms of the executeRequestResponseAsync method:

```
public WSIFCorrelationId executeRequestResponseAsync
    (WSIFMessage input, WSIFResponseHandler handler)
```

and

```
public WSIFCorrelationId executeRequestResponseAsync (WSIFMessage input)
```

The first of these takes an input message and a **WSIFResponseHandler**. The WSIFResponseHandler is invoked on another thread when the operation completes. When using this method the client listener would call the **fireAsyncResponse** method which will call the WSIFResponseHandler **executeAsyncResponse** method.

Here is the Javadoc for the WSIFResponseHandler interface.

The other form of **executeRequestResponseAsync** that only takes a WSIFMessage input message does not use a response handler, and the client listener should process the response by calling the WSIFOperation method **processAsyncResponse**. This will update the WSIFMessage output and fault messages with the result of the request.

WSIF supports correlation between the request and response in the asynchronous request-response case. When the request is sent, the WSIFOperation is serialized into the **WSIFCorrelationService**. The **executeRequestResponseAsync** methods returns a **WSIFCorrelationId** object which identifies the serialized WSIFOperation. The client listener can use this to match a response to a particular request.

The correlation service is located with the **getCorrelationService()** method of the **WSIFCorrelationServiceLocator** class in the org.apache.wsif.utils package.

In a managed container a default correlation service is defined in the default Java Naming and Directory Interface (JNDI) namespace using the name: java:comp/wsif/WSIFCorrelationService . If this is not available, then WSIF will use the WSIFDefaultCorrelationService.

Here is the Javadoc for the WSIFCorrelationService interface.

and this is the correlator id:

```

public interface WSIFCorrelator extends Serializable {
    public String getCorrelationId();
}

```

The client must implement their own response message “listener” or MDB in order to recognize the arrival of response messages themselves. This would manage the correlation of the response message to the request and call of one of the asynchronous response processing methods. As an example of the type of thing required of a client listener, the following code fragment shows what could be in the **onMessage** method of a Java Messaging Service (JMS) listener:

```

public void onMessage(Message msg) {
    WSIFCorrelationService cs =
        WSIFCorrelationServiceLocator.getCorrelationService();
    WSIFCorrelationId cid =
        new JmsCorrelationId( msg.getJMSCorrelationID() );
    WSIFOperation op = cs.get( cid );
    op.fireAsyncResponse( msg );
}

```

WSIFOperation - Synchronous and asynchronous timeouts reference:

When using the Web Services Invocation Framework (WSIF) with JMS there are timeouts that can be set for synchronous and asynchronous operations.

Default values for these timeouts are defined in the WSIF properties file as follows:

```

# maximum number of milliseconds to wait for a response to a synchronous
# request. Default value if not defined is to wait forever.
wsif.syncrequest.timeout=10000

# maximum number of seconds to wait for a response to an async request.
# if not defined on invalid defaults to no timeout
wsif.asyncrequest.timeout=60

```

The defaults mean a synchronous request (such as a WSIFOperation **executeRequestResponseOperation** method call) will timeout after ten seconds, and an asynchronous request (such as a WSIFOperation **executeRequestResponseAsync** method call) will timeout after sixty seconds.

Note:

The WSIFProperties class **getAsyncTimeout** method multiplies the `wsif.asyncrequest.timeout` value by 1000. This converts the value from seconds in the properties file, to milliseconds used for the timeout value.

These default values can be overridden for a given request by setting a JMS property on the operation request with the `jms:property` and `jms:propertyValue` WSDL elements. The name of the property to set should be the name of the timeout from the WSIF properties file.

For example the following sets synchronous requests to timeout after two minutes (120 seconds). These sets are shown on multiple lines for publication:

```

<jms:propertyValue name=
    "wsif.syncrequest.timeout" type="xsd:string" value="120000"/>

```

and the following line disables asynchronous timeouts (a value of zero means wait for ever):

```

<jms:propertyValue name="wsif.asyncrequest.timeout" type=
    "xsd:string" value="0"/>

```

Note: When an asynchronous timeout expires, no listener or MDB waiting for the response is notified. The asynchronous timeout is only used to tell the correlation service that the stored WSIFOperation may be deleted.

WSIF: Resources for learning

Use these links to find supplementary information about the Web Services Invocation Framework (WSIF).

This information resides on IBM and non-IBM Internet sites, and in each case the site sponsor controls the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

- WebSphere Version 5 Web services Handbook. This Redbook illustrates with suitable demonstration applications how Web services can be implemented using the IBM product portfolio, especially WebSphere Application Server Version 5 and WebSphere Studio Application Developer Version 5. It includes a chapter on WSIF.
- SOAP. This article is a detailed overview of SOAP, which includes programming specifications.
- Web Services Description Language. This article is a detailed overview of Web services Description Language (WSDL), which includes programming specifications.
- The Apache Software Foundation. The Apache Software Foundation provides support for the Apache community of open-source software projects. Of particular interest is the Apache Web services project. The WSIF source code has been donated by IBM to the Apache Software Foundation, and is maintained here as an Apache project.
- JSR109. This is the Java Community Process specification for implementing Web services in Java.

See also the set of Web services links given in Web services: Resources for learning.

Chapter 8. IBM WebSphere UDDI Registry

Welcome to the IBM WebSphere UDDI Registry.

Use the table of contents (on the left and below) to view the various topics for a specific product or technology. Select the topic you are interested in to either open documentation locally or find information about how to locate documentation.

- Terminology
- Definitions
- Overview of UDDI Registries
- Migrating from the IBM WebSphere UDDI Registry on WebSphere Application Server 4.0
- Installing the UDDI Registry Component
- **5.0.1 +** Applying Service to the UDDI Registry in a Network Deployment and single Application Server environment
- Use of a remote DB2 Database
- Reinstalling the UDDI Registry Component
- Removing the UDDI Registry application from a deployment manager cell
- Removing the UDDI Registry application from a single appserver
- Configuring the UDDI Registry
- Administering the UDDI Registry
- The UDDI user console
- The SOAP Application Programming Interface
- The application programming interface
- The EJB Interface
- Problem determination
- Messages
- Samples
- Installation Verification Program (IVP)
- Reporting Problems with the IBM WebSphere UDDI Registry
- Feedback

UDDI Registry terminology

When reference is made to the directory location of the WebSphere Application Server it is referred to as **<WebSphere-install-dir>** and , the directory location of the WebSphere Deployment manager as **<DeploymentManager-install-dir>**. These translate, by default, to the following locations:

Windows

<WebSphere-install-dir>

C:\Progra~1\WebSphere\AppServer\

<DeploymentManager-install-dir>

C:\Progra~1\WebSphere\DeploymentManager\

5.0 + Linux/Solaris Platforms

5.0.1 + Linux/Solaris/HP Platforms

<WebSphere-install-dir>

/opt/WebSphere/AppServer/

<DeploymentManager-install-dir>

/opt/WebSphere/DeploymentManager/

AIX Platform

<WebSphere-install-dir>

/usr/WebSphere/AppServer/

<DeploymentManager-install-dir>

/usr/WebSphere/DeploymentManager/

z/OS Platform

<WebSphere-install-dir>

/WebSphere390/V5R0M0/AppServer/

<DeploymentManager-install-dir>

/WebSphere390/V5R0M0/DeploymentManager/

UDDI Registry definitions

bindingTemplate

Technical information about a service entry point and construction specifications.

businessEntity

Information about the party who publishes information about a family of services.

businessService

Descriptive information about a particular service.

publisherAssertion

Information about a relationship between two parties, asserted by one or both.

tModel

Short for technical model.

A tModel is a data structure representing a reusable concept, such as a Web service type, a protocol used by Web services, or a category system.

tModel keys within a service description are a technical fingerprint that can be used to trace the compatibility origins of a given service. They provide a common point of reference that allows compatible services to be easily identified.

tModels are used to establish the existence of a variety of concepts and to point to their technical definitions. tModels that represent value sets such as category, identifier, and relationship systems are used to provide additional data to the UDDI core entities to facilitate discovery along a number of dimensions. This additional data is captured in keyedReferences that reside in category Bags, identifierBags, or publisherAssertions. The tModelKey attributes in these keyedReferences refer to the value set that related to the concept or namespace being represented. The keyValues contain the actual values from that value set. In some cases keyNames are significant, such as for describing relationships and when using the general keywords value set. In all other cases, however, keyNames are used to provide a human readable version of what is in the keyValue.

An overview of IBM UDDI Registries

The Universal Description, Discovery and Integration (UDDI) specification defines a way to publish and discover information about Web Services. The term 'Web service' describes specific business functionality exposed by a company, usually through an Internet connection, to allow another company, or its subsidiaries, or software program to use the service.

Universal Business Registries (IBM UBR)

The IBM Universal Business Registry is one of a group of web-based registries that expose information about a business or other entity and its technical interfaces (or APIs). These registries are run by multiple Operator Sites, and can be used by anyone who wants to make information available about one or more businesses or entities, as well as anyone that wants to find that information. Although there are Universal Business Registries (sometimes referred to as 'public UDDI registries') hosted worldwide, including one hosted by IBM, enterprises may wish to host their own internal registries behind their firewall to better manage their internal implementation of Web Services.

For more detailed information about UDDI in general visit <http://www.uddi.org>

IBM WebSphere UDDI Registry

The IBM WebSphere UDDI Registry is a directory for Web Services that is implemented using the UDDI specifications. In contrast with the IBM UBR, this component of WebSphere Network Deployment is a product offering for companies or industries to implement.

A critical component of IBM's dynamic e-business infrastructure, IBM WebSphere UDDI Registry solves the problem of discovery of technical components for an enterprise and its partners by:

- Providing control, flexibility and confidentiality so that an enterprise can protect its e-business investments
- Increasing efficiency by making it easier to identify technical assets
- Leveraging existing infrastructures

For example, the IBM WebSphere UDDI Registry could be used in the following way within a large enterprise:

A company has a legacy application that provides phone numbers and Human Resources (HR) information of employees. This is turned into a Web Service and published to the registry. A developer in the same company needs to write an application for a procurement function that also needs to provide HR information to the supplier. The application should allow the supplier to have access to the employee account codes once the employee provides his name or serial number. Before Web Services, the developer had three choices:

1. Would not have known about the similar application
2. Knew about it but could not reuse due to technical barriers
3. Knew about it and reused only after significant time and negotiation

With UDDI, the developer can search for the "web service" and reuse the existing technical component in his new application for the supplier in a matter of minutes. The developer saves time and gets the application up and running sooner than he would have otherwise -- increasing efficiency and saving the company time and money. The IBM WebSphere UDDI Registry is the first version 2 standard-compliant UDDI registry for private enterprise work. The IBM WebSphere UDDI Registry:

- Supports the public UDDI V2.0 standard
- Leverages the proven, reliable WebSphere Application Server technology
- Uses a relational database, such as DB2, for its persistent store.

Installing and setting up a UDDI Registry

If you wish to use the UDDI User Console using Internet Explorer as your web browser, and using SSL, then you must use Internet Explorer V5.5 with SP2 and security fix Q321232 (which must be applied in that order), or later.

If you are migrating from the IBM WebSphere UDDI Registry product that was available to run on WebSphere Application Server 4.0, then you need to read the section Migrating from the IBM WebSphere UDDI Registry on WebSphere Application Server 4.0 before installing the new product.

Choice of database product to be used as the persistence store

The UDDI Registry application can use either DB2 or Cloudscape as the persistence store for the registry data.

- If you plan to use the UDDI Registry in production then you must use DB2 as your persistence store.
- If you plan to use the UDDI Registry for development and testing purposes, then you can choose to use Cloudscape as your persistence store. Please note that Cloudscape is not intended for production purposes.

The UDDI Registry application can use either DB2 or Cloudscape as the persistence store for the registry data. However, in a z/OS environment, you should use DB2. If you use Cloudscape, you must restrict the WLM policy to the server to allow only one servant process.

Steps for this task

You are given the option to install the UDDI Registry as part of the IBM WebSphere Application Server for z/OS ISPF Customization Dialog. See *WebSphere Application Server for z/OS V5 Installation and Customization* for more information on how to install the UDDI Registry. The latest version of this publication is available on the product library page at URL http://www.ibm.com/software/webservers/appserv/zos_os390/library.html

As part of the installation of the IBM WebSphere Application Server with Network Deployment option, you are given the option to install the "UDDI Registry", which is shown under Web Services. Having selected the UDDI Registry and installed the various files that make up the application, you have two choices as to the environment in which you run it:

1. Install the UDDI Registry application into the deployment manager cell using DB2 or Cloudscape as the database in which the UDDI Registry data will be held, selecting one of the application servers within the cell in which to run the UDDI Registry.
2. Install the UDDI Registry application directly into an application server using DB2 or Cloudscape as the database in which the UDDI Registry data will be held. Please note that, if you choose this option, then the application server on which you run the UDDI Registry must not be added into a deployment manager cell, as this would cause the file synchronization within the cell to remove the application.

In most cases you will probably choose option 1, and install the UDDI Registry into a deployment manager cell, but you might find that option 2, to install the UDDI Registry into a stand-alone application server, is useful for development or test purposes.

Note:

- Several WebSphere commands are used during the following procedures, some of which must execute on the DeploymentManager and some of which must execute on the target Application server. The instructions below will distinguish which is appropriate for each command. The WebSphere commands will be found in the bin subdirectory of the appropriate WebSphere install tree. In order to ensure correct operation of these commands, you will need to do one of the following:
 - Ensure that the appropriate bin subdirectory is in your path prior to executing the command
 - Change directory to the appropriate bin subdirectory
 - Fully qualify the path to the commands
- It is recommended that you use the version of java shipped with WebSphere found in the java/bin subdirectory.

The following table lists the UDDI Registry files, and the locations into which they are placed by the install. The “Location” column shows the subdirectory under the WebSphere DeploymentManager install directory. For example, if you had installed IBM WebSphere Application Server with Network Deployment option onto a machine running Windows, and had used the default directory, then a location of “installableApps” would mean that the file had been placed into the C:\Program Files\IBM\WebSphere\DeploymentManager\installableApps directory. For Windows platforms, read the “/” directory separator in the location column as a “\” directory separation character.

Files	Purpose	Location
uddi.ear	The UDDI Registry application itself, which is packaged and runs as an Enterprise Application	installableApps
uddi.properties	Provides configuration properties for the UDDI Registry application	properties
uddiresourcebundles.jar	Contains system messages for the UDDI Registry application	lib
uddicloudscapeuserfunc.jar	Contains functions that are used by Cloudscape if the Cloudscape database is used with the UDDI Registry	lib
setupuddi.jacl	Admin script to create a JDBC driver and datasource for the UDDI Registry, and to install the UDDI Registry application in a DeploymentManager Cell	UDDIReg/scripts
setupuddimessages.jar	Contains setup and install messages for the UDDI Registry application	lib
removeuddi.jacl	Admin script to undo the effects of setupuddi.jacl	UDDIReg/scripts
appserverremoveuddi.jacl	Admin script to undo the effects of appserversetupuddi.jacl	UDDIReg/scripts
appserversetupuddi.jacl	Admin script to create a JDBC driver and datasource for the UDDI Registry, and to install the UDDI Registry application in a single, stand-alone, application server	UDDIReg/scripts
SetupDB2UDDI.jar	The ‘UDDI DB2 Setup Wizard’, to create and pre-load the UDDI Registry database if DB2 is to be used as the persistence store	UDDIReg/scripts
UDDI20 (directory)	Cloudscape directory containing the UDDI Registry tables and pre-loaded data	bin

uddiejbclient.jar	Class library for use when writing an EJB client to access the UDDI Registry	UDDIReg/ejb
Various javadoc files	JAVADOC to describe the EJB interface to the UDDI Registry	UDDIReg/ejb/javadoc
UDDITaxonomyTools.jar	Provides tools for supporting custom taxonomies with the UDDI Registry	UDDIReg/scripts
CustomTaxonomy.properties	Provides configuration properties to be used the the UDDITaxonomyTools	UDDIReg/scripts

If you intend to run in a Deployment Manager Cell then complete the following task - **Installing the UDDI Registry into a deployment manager cell**

If you intend to run in a single WebSphere Application server, then complete the following task - **Installing the UDDI Registry into a single WebSphere Application Server**

Continue with Configuring the UDDI Registry.

Installing the UDDI Registry into a deployment manager cell

These instructions assume that the installation has been performed into a clean environment. If you are installing into an existing Deployment Manager cell, then steps 1 to 5 must be omitted, i.e. skip to bullet 6 immediately.

1. Install the WebSphere Application Server for Network Deployment package, and select the UDDI Registry option under Web Services. Install the WebSphere Application Server for z/OS product.
2. Install one or more base application servers which will form the cell of servers. One of these should be the application server in which you plan to run an instance of the UDDI Registry. You can run more than one instance of the UDDI Registry within a cell of servers: the UDDI Registry application name will be suffixed with the target node and server names to make it unique within the cell (See also "Advanced use of setupuddi.jacl"), but you can only run one UDDI instance within each application server.
3. Ensure that the target application server is stopped.
4. Run startManager (startManager.sh on Unix Platforms) on the deployment manager node to start the deployment manager. Issue the following command from the install_root/bin directory:

```
START dmgr_proc_name, JOBNAME=server_short_name,
      ENV=cell_short_name.node_short_name.server_short_name
```

Note: This command must be entered on a single line. It is split here for display purposes.

On Unixz/OS platforms you must remember to run the db2profile script before issuing the STARTstartManager.sh command. This script is located within the DB2 instance's home directory under SQLLIB and can be invoked, for example, by typing:

```
". /home/db2inst1/sqllib/db2profile"
```

5. Run addNode (addNode.sh on Unix Platforms) on each of the base application server(s) to add it as a node into the cell. (How to run addNode is described elsewhere in the InfoCenter). For example, addnode myriad - where myriad is the name of your deployment manager host.

6. Copy the *uddiejbclient.jar* file and the EJB javadoc directory tree from the UDDIReg/ejb subdirectory of the deployment manager install tree onto any machine(s) where you will be creating EJB clients to access the UDDI Registry.
7. If you have any global configuration properties that will be common to any UDDI Registries that you install into this cell, then you can edit the *uddi.properties* file in the properties subdirectory of the deployment manager install tree to set them up. (See the section on Configuring global UDDI properties for more details about the global configuration properties).
8. The UDDI Registry application is supplied with the security permissions that it requires to execute. This step explains how you can see the permissions that have been set, and change them if you feel that it is appropriate to do so. *It is recommended that you only do this if you have a thorough understanding of Java 2 security issues, and the way in which security permissions are used by WebSphere.* The permissions for the UDDI Registry application are set within a file *was.policy* which is part of the *uddi.ear* application file. To see and change the contents of this file you should:
 - a. On the deployment manager, copy the **uddi.ear** file from the installableApps subdirectory of the deployment manager install tree into a temporary directory.
 - b. Un-jar the *uddi.ear* file (i.e. unpack *uddi.ear* using the 'jar -x' command).

For example,

```
jar -x uddi.ear
```

(This uses the jar command in the bin subdirectory of the deployment manager, so you might need to fully qualify the path to the jar command.)

- c. You will find the *was.policy* file under the META-INF subdirectory that is created.

This will allow you to see the permissions which have been granted to the UDDI Registry application, and to make any changes that are necessary. Please note that if you make any errors in changing this file, then the UDDI Registry application might either fail to start, or will encounter errors when trying to execute UDDI requests.

- d. Re-jar the *uddi.ear* file using the jar command.

For example,

```
jar -cf uddi.ear .
```

Note: NOTE the space and the dot after *uddi.ear*)

(This uses the jar command in the <install_root>\java\bin subdirectory of the deployment manager, so you might need to fully qualify the path to the jar command.)

- e. Copy the new *uddi.ear* back to the installableApps directory.
9. Please note that if the target application server is running, then this step will stop and restart it. If you are planning to use Cloudscape for the database in which the UDDI Registry will be held, please read the section "Setting up the UDDI Registry to use Cloudscape within a deployment manager cell" and then return to this point. If however, you plan to use DB2, then please refer to the section "Setting up the UDDI Registry to use DB2 within a deployment manager cell" and then return to this point.
10. Ensure that the UDDI Registry is configured appropriately for your installation, as described in the section on Configuring the UDDI Registry.

11. Start, or stop and restart, the target application server. This should also start the UDDI Registry application. If not, use the admin console on the deployment manager to do so.

Advanced use of setupuddi.jacl

A number of symbols are defined at the top of the setupuddi.jacl script. These allow you to control the amount of logging that is performed, and to install multiple instances of the UDDI Registry within the same cell.

The symbols which you can edit are as follows:

- **logEnabled** - default setting is 1 which causes the progress of the script to be logged. Setting this symbol to 0 causes information logging to be suppressed, with only error messages being output.
- **overwriteExisting** - default setting is 1 which causes any existing installation of the UDDI Registry application to be overwritten. Setting this symbol to 0 would cause the existing installation to be left as is, but would allow other files used by the UDDI Registry to be updated. You are recommended to only change this setting under the guidance of IBM Service.
- **appName** - default setting is UDDIRegistry, which will be the first part of the name used for the UDDI Registry application installed into the target server. To ensure uniqueness of application names within the cell, the full application name that will be used is <appName>.<nodeName>.<server>, where <nodeName> is the name of the target node and <server> is the name of the target server. You can choose to change the first part of this (the <appName>) portion by changing the setting of this symbol before running setupuddi.jacl, although it is generally recommended that you do not change this value.

Continue with Configuring the UDDI Registry.

Setting up the UDDI Registry to use Cloudscape within a deployment manager cell

If you plan to use Cloudscape for the database in which the UDDI Registry data will be held, use this task to create and load the UDDI Registry database using Cloudscape.

Please see the section "Choice of database product to be used as the persistence store" to decide which database product you should use as your persistence store before proceeding further with this step.

This task is part of the parent task and is described in Installing the UDDI Registry into a deployment manager cell. You should complete this task at the appropriate step in the parent task, then return to and complete the parent task.

If you plan to use Cloudscape for the database in which the UDDI Registry data will be held, use this task to setup and install the UDDI Registry application to use the supplied Cloudscape database.

This task configures Cloudscape on the host where you want to run the UDDI Registry. Cloudscape is supplied with WebSphere Application Server, so you should not need to install Cloudscape support.

In this task you will invoke a script called setupuddi.jacl, specifying the target node and application server into which the UDDI Registry is to be deployed.

Please note that if the target application server is running when you invoke `setupuddi.jacl`, then the script will stop the server and will restart the server after it has completed its operations.

1. Copy the UDDI20 directory tree from the bin subdirectory of the deployment manager tree into the bin subdirectory of the target application server's install tree.
2. Create a JDBC driver and datasource to provide access to the UDDI20 Cloudscape database, and install the UDDI Registry application. This is done using the WSADMIN tool, using as input the `setupuddi.jacl` script from the `UDDIReg/scripts` subdirectory of the Deployment Manager. Note that this script must be run on the deployment manager node.

You should either run this script from the `UDDIReg/scripts` subdirectory where it is located, or copy it to some other suitable directory. Note that the WSADMIN command is located in the bin subdirectory of the deployment manager node. The syntax for calling this script for Cloudscape is:

```
wsadmin -f setupuddi.jacl
        deploymgrpath
        servername
        nodename
        discoveryURLprefix
        pathtodb
        > setupuddi.log
```

where

- *deploymgrpath* is the fully qualified pathname of the deployment manager install directory, specified using forward slashes regardless of platform; e.g. for Windows, this might be `C:/Progra~1/WebSphere/deploymentManager`
- *servername* is the name of the target server on which you wish to deploy the UDDI Registry, such as `server1`. Note the the server name entered is case sensitive.
- *nodename* is the name of the WebSphere node on which the target server runs. Note the the node name entered is case sensitive.
- *discoveryURLprefix* is the URL prefix to be used for discovery URLs. Typically this will be of the form `http://<ip-address>:9080/uddisoap/` so an example of a *discoveryURLprefix* value might be `http://mynode.mylocation.mycompany.com:9080/uddisoap/`
- *pathtodb* is the path to the UDDI20 database within the bin subdirectory of your WebSphere AppServer installation, specified using forward slashes regardless of platform; e.g. for Windows, this might be `C:/Progra~1/WebSphere/AppServer/bin/UDDI20` and for Unix platforms, it might be `/opt/WebSphere/AppServer/bin/UDDI20`
- `> setupuddi.log` is an optional parameter to direct the output to a log file as opposed to the default (which is to the screen)

For example (shown here on multiple lines for publication)::

```
wsadmin -f setupuddi.jacl "C:/Progra~1/WebSphere/DeploymentManager/" server1
MYRIAD "http://myriad.headoffice.xyz.com:9080/uddisoap/"
"C:/Progra~1/WebSphere/Appserver/bin/UDDI20"
```

will install the UDDI Registry application into the server `server1` running on node `MYRIAD`, and set it up to access the Cloudscape UDDI20 database located in the bin subdirectory of the application server.

The `setupuddi.jacl` script will

- a. Create a JDBC driver named `UDDI.JDBC.Driver.<nodeName>.<server>` and a datasource named `UDDI.Datasource.<nodeName>.<server>` (where

- <nodeName> is the name of the target node and <server> is the name of the target server>, and will replace any existing driver and datasource of that name.
- b. Check whether the UDDI Registry application is already installed and, if so, stop it and uninstall it.
 - c. Update the uddi.properties configuration property file to configure the discoveryURLprefix value that you have specified and set the persister property as 'Cloudscape', and place the uddi.properties file into the location config/cells/<currentcell>/nodes/<nodename>/servers/<servername>. You should make any further global configuration changes using this copy of the file.
 - d. Place a number of files that are needed by the UDDI Registry into the WebSphere configuration repository, and update the ws.ext.dirs list to reference these files.
 - e. Install the UDDI Registry.

This script will deploy the UDDI Registry into the configuration under the deployment manager, and then do a Synch which causes it to get installed into the specified server.

Note: The setup script, setupuddi.jacl, cannot be used to install the UDDI Registry application into a clustered application server. It is possible to cluster the UDDI Registry application by installing UDDI into an unclustered application server using the setup script, and then clustering that application server.

Return to the next step in the parent task Installing the UDDI Registry into a deployment manager cell.

Setting up the UDDI Registry to use DB2 within a deployment manager cell

Please see the section "Choice of database product to be used as the persistence store" to decide which database product you should use as your persistence store before proceeding further with this step.

This task is part of the parent task and is described in Installing the UDDI Registry into a deployment manager cell. You should complete this task at the appropriate step in the parent task, then return to and complete the parent task.

If you plan to use DB2 for the database in which the UDDI Registry will be held, ensure that the correct prerequisite fix packs have been applied as listed at <http://www-3.ibm.com/software/webservers/appserv/doc/latest/prereq.html> otherwise the startup of the UDDI DB2 setup wizard will fail.

If you plan to use DB2 for the database in which the UDDI Registry data will be held, use this task to create and load the UDDI Registry database using DB2, and to setup and install the UDDI Registry application to use the DB2 database.

This task uses the UDDI DB2 Setup Wizard to configure DB2 on the system where you want to run the UDDI Registry. Before starting this task, ensure that DB2 is installed and running on that system.

Copy the UDDIReg directory tree from the deployment manager to the target application server where DB2 will run.

The following steps should be carried out on the system on which the target application server is located (referred to below as the 'target system').

In this task you will invoke a script called `setupuddi.jacl`, specifying the target node and application server into which the UDDI Registry is to be deployed. Please note that if the target application server is running when you invoke `setupuddi.jacl`, then the script will stop the server and will restart the server after it has completed its operations.

1. On Windows, ensure that since installing DB2 you have run the `usejdbc2.bat` command file. For more information about this, see "Application Building Guide" in the DB2 documentation.
2. Create and load the UDDI Registry database, called UDDI20.

Note: If you are migrating from an earlier version of the UDDI Registry, and your UDDI20 DB2 database already exists, then you should skip this step unless you want to replace the existing database with a new UDDI20 DB2 database. If you do choose to replace an existing database then all of your existing UDDI data will be lost. **Important:** please also note that if you do choose to replace the existing UDDI20 database, then there must not be any applications or users accessing the database at the time that you run the UDDI DB2 setup wizard.

If you are planning to use a remote DB2 system on another host machine, you will need to copy the `SetupDB2UDDI.jar` file to the remote system and run it on that system to create and load the UDDI Registry database following the instructions within this bullet and continue with the next bullet (which states "If using a remote DB2 system on another host machine..") on the local host.

The UDDI DB2 Setup Wizard used in this task will prompt you to provide the DB2 userid and password under which the UDDI Registry database will be created and subsequently accessed. Before starting this task, ensure that you have created an appropriate DB2 userid and password. This same userid and password must be used throughout the following steps where the DB2 userid and password is requested.

5.0.1 + If you are using a non-English installation of DB2, then please ensure you have applied PTF1 before continuing.

Information on how to do this and where to obtain the PTF can be found [here](#).

5.0.1 + If you wish to implement the updated `SetupDB2UDDI.jacl` database wizard, then also please the PTF(s) mentioned above.

Note: **5.0.1 +** If you have a copy of the file `SetupDB2UDDI.jar` in your appserver directory, then the application of the base and Network Deployment PTFs will not update `SetupDB2UDDI.jar` in your appserver directory. You must apply the PTF for Network Deployment to your `DeploymentManager` file structure to update the `SetupDB2UDDI.jar` located there (in the `/UDDIReg/scripts` subdirectory), and then manually copy this jar to any appserver you may wish to run it on.

On Windows this should be a userid and password with administrative privileges. On Unix platforms, you should supply the userid and password of the DB2 instance in which you wish the database to be created.

To create the database you use the UDDI DB2 setup wizard, which is supplied as a shell script called SetupDB2UDDI.sh in the UDDIReg/binjar file called SetupDB2UDDI.jar in the UDDIReg/scripts subdirectory, by following these steps:

- a. Change directory to the directory containing the file SetupDB2UDDI.jar (that is, either the UDDIReg/scripts directory in which it is supplied, or a directory on the target system into which you have copied it).
- b. In order to run the wizard, you need to first ensure that you have access from your command line to the JVM supplied with WebSphere. This is done as follows:

- On Windows, on a user with administrative privileges, in a command window type the following command:

```
was_install\bin\setupcmdline.bat
```

Where *was_install* is the path to the directory where you installed WebSphere Application Server.

For example,

```
C:\Program Files\WebSphere\AppServer\bin\setupcmdline.bat
```

- On Unix platforms, at a command line type one of the following commands:

- If you are using bash, then as the root user run

```
./opt/WebSphere/AppServer/bin/setupCmdLine.sh
```

- If you are using csh, then as the root user run

```
source /opt/WebSphere/AppServer/bin/setupCmdLine.sh
```

where */opt/WebSphere/AppServer* is the path to the directory where you installed WebSphere Application Server.

- c. In the same command window, start the UDDI DB2 setup wizard by typing one of the following commands:

- To start a graphical user interface, type

```
java -jar SetupDB2UDDI.jar
```

- To start a text mode interface, type:

```
java -jar SetupDB2UDDI.jar -console
```

Note: The above commands will fail unless your DB2 installation has the correct fix packs applied as listed on the prerequisite page found at <http://www-3.ibm.com/software/webservers/appserv/doc/latest/prereq.html>

- d. Follow the prompts to work through the wizard panels.
- e. If necessary, check the log files for the wizard. A log file called UDDIloadDB.log is written to the directory from which the wizard is run (but note that on Windows platforms, if you have decided not to overwrite an existing UDDI20 database, then this fact will not be logged, and the log file will not be created).
- f. Gather the following information about your DB2 z/OS configuration from your DB2 administrator:
 - 1) The location of the DB2 system where you want to persist the UDDI directory.
 - 2) The name of the DB2 database that will be used for the UDDI directory. If a database does not already exist, the UDDI DB2 setup wizard will create one for you.

- 3) The name of the DB2 tablespace that will be used for the UDDI directory. If a tablespace does not already exist, the UDDI DB2 setup wizard will create one for you.
 - 4) The name of the DB2 storage group that will hold the UDDI directory.
 - 5) The name of the volume on which the DB2 database should be created.
 - 6) The name of the alias of the VSAM catalog for DB2 datasets.
- g. Run the UDDI DB2 setup wizard

The UDDI DB2 setup wizard supports two modes of operation:

1) Create and load

To initiate this mode of operation, issue the following command from an MVS command line prompt:

```
/install_root/UDDIReg/bin/SetupDB2UDDI.sh DB2_location_name
DB2_database_name DB2_tablespace_name create
DB2_storage_group_name volume_name
VSAM_catalog_name
```

This command should be entered all on a single line. It was split here for formatting purposes.

This mode of operation requires you to specify all of the parameters on the shell script. This enables the DB2 database and tablespace to be created. The newly created database will be primed for use by the UDDI Registry.

Note: If you use the create and load mode of operation, you must run the wizard using a USERID that has the authority to create or update DB2 databases, storage groups, ect. Also, that USERID will be used as the SQLID of the resulting DB2 tables.

2) Load

To initiate this mode of operation, issue the following command from an OMVS command line prompt:

```
/install_root/UDDIReg/bin/SetupDB2UDDI.sh DB2_location_name
DB2_database_name DB2_tablespace_name skip
```

This command should be entered all on a single line. It was split here for formatting purposes.

This mode of operation does not require you to specify the *DB2_storage_group_name volume_name* and *VSAM_catalog_name* parameters on the shell script. This mode uses the existing DB2 database and tablespace and only primes that database for use by the UDDI Registry.

The output from the UDDI DB2 wizard will be placed in a file called *setupDB2UDDI.tracelog*. Errors that occur will be written to the user's telnet session.

3. If using a remote DB2 system on another host machine, refer to "Use of a remote DB2 database" and then return to this point and continue with the instructions below.
4. Create a JDBC driver and datasource to provide access to the UDDI20 DB2 database, and install the UDDI Registry application. This is done using the WASADMIN tool, using as input the *setupuddi.jacl* script from the UDDIReg/scripts subdirectory of the Deployment Manager. Note that this script must be run on the deployment manager node.

You should either run this script from the UDDIReg/scripts subdirectory where it is located, or copy it to some other suitable directory. Note that the WSADMIN command is located in the bin subdirectory of the deployment manager node. The syntax for this script for DB2 is:

```
wsadmin -f setupuddi.jacl
        deploymgrpath
        servername
        nodename
        discoveryURLprefix
        dbname
        db2userid
        db2password
        db2ziplocation
        > setupuddi.log
```

where

- *deploymgrpath* is the fully qualified pathname of the deployment manager install directory, specified using forward slashes regardless of platform; e.g. for Windows, this might be c:/Progra~1/WebSphere/DeploymentManager
- *servername* is the name of the target application server on which you wish to deploy the UDDI Registry, such as server1. Note that the server name is case sensitive.
- *nodename* is the name of the WebSphere node on which the target application server runs. Typically, this will be the same as the machine name. Note that the node name is case sensitive.
- *discoveryURLprefix* is the URL prefix to be used for discovery URLs. Typically this will be of the form http://<ip-address>:9080/uddisoap/ so an example of a discoveryURLprefix value might be http://mynode.mylocation.mycompany.com:9080/uddisoap/
- *dbname* is the name of the UDDI Registry database under DB2. For this parameter, you should specify UDDI20the database name you specified when you ran the UDDI DB2 setup wizard.

Note: If a remote DB2 system is being used the *dbname* stated here must be the alias created when access to the remote database was set up on the local system

- *db2userid* and *db2password* are a valid DB2 userid and password with administrative privileges
- *db2ziplocation* is the path to the db2java zip file on your system, specified using forward slashes regardless of platform; e.g. for Windows, this might be C:/Progra~1/SQLLIB/java/db2java.zip
- > *setupuddi.log* is an optional parameter to direct the output to a log file as opposed to the default (which is to the screen)

For example (shown here on multiple lines for publication):

On Windows the command is:

```
wsadmin -f setupuddi.jacl "C:/Progra~1/WebSphere/deploymentManager/" server1
MYRIAD "http://myriad.headoffice.xyz.com:9080/uddisoap/" UDDI20 db2admin
secretpwd "C:/Progra~1/SQLLIB/java/db2java.zip" > setupuddi.log
```

On Unix platforms the command is:

```
wsadmin.sh -f setupuddi.jacl "/opt/WebSphere/deploymentManager/" server1
MYRIAD "http://myriad.headoffice.xyz.com:9080/uddisoap/" UDDI20 db2admin
secretpwd "/home/db2inst1/sqllib/java12/db2java.zip" > setupuddi.log
```

will install the UDDI Registry application into the server server1 running on node MYRIAD, and set it up to access the DB2 UDDI20 database using the userid 'db2admin' and password 'secretpwd'.

The setupuddi.jacl script will:

- a. Create a JDBC driver named UDDI.JDBC.Driver.<nodeName>.<server> and a datasource named UDDI.Datasource.<nodeName>.<server> (where <nodeName> is the name of the target node and <server> is the name of the target server), and will replace any existing driver and datasource of that name.
- b. Check whether the UDDI Registry application is already installed and, if so, stop it and uninstall it.
- c. Update the uddi.properties configuration file to configure the discoveryURLprefix value that you have specified, and to set the persister property as 'DB2', and place the uddi.properties file into the location config/cells/<currentcell>/nodes/<nodename>/servers/<servername>. You should make any further global configuration changes using this copy of the file.
- d. Place a number of files that are needed by the UDDI Registry into the WebSphere configuration repository, and update the ws.ext.dirs list to reference these files.
- e. Install the UDDI Registry application into the server server1 running on node MYRIAD, and set it up to access the DB2 database..

Note: The setup script, setupuddi.jacl, cannot be used to install the UDDI Registry application into a clustered application server. It is possible to cluster the UDDI Registry application by installing UDDI into an unclustered application server using the setup script, and then clustering that application server.

Return to the next step in the parent task Installing the UDDI Registry into a deployment manager cell.

Installing the UDDI Registry into a single appserver

If you intend to run in a single WebSphere Application server, then complete the following task.

When you select the UDDI Registry option, then the installation will place all files that are needed to run a UDDI Registry onto the deployment manager install tree on the machine on which you are installing IBM WebSphere Application Server for Network Deployment.

To be able to run the UDDI Registry in a single application server instance in your network space you must copy these files over to the application server and then deploy the UDDI Registry. You can do this as follows:

1. Stop the application server on which you plan to run the UDDI Registry; for example, using the command `stopServer server_name` (`stopServer.sh` on Unix Platforms) by issuing the following command from the bin directory:

```
STOP appserver_proc_name,JOBNAME=server_short_name1,  
    ENV=cell_short_name.node_short_name.server_short_name
```

Note: This command must be entered on a single line. It is split here for display purposes.

2. Copy the *uddi.ear* file from the installableApps subdirectory of the deployment manager install tree into the installableApps subdirectory of the target application server's install tree.
3. Copy the *uddi.properties* file from the properties subdirectory of the deployment manager install tree into the properties subdirectory of the target application

server's install tree. In a subsequent step, you configure the UDDI Registry using the properties in the *uddi.properties* file.

4. Copy both the *uddiresourcebundles.jar* and the *setupuddimessages.jar* files from the lib subdirectory of the deployment manager install tree into the lib subdirectory of the target application server's install tree.
5. Optionally, if you are going to write or run code that uses the EJB interface to UDDI on another machine, then copy the *uddiejbclient.jar* file and the EJB javadoc directory tree from the UDDIReg/ejb subdirectory of the deployment manager install tree onto a location of your choice on any machines where you will be creating EJB clients to access the UDDI Registry.
6. Configure database support for the UDDI Registry database, in which the UDDI Registry will be held. To do this, complete one of the following tasks then return this point:
 - Setting up the UDDI Registry to use Cloudscape in a single AppServer
 - Setting up the UDDI Registry to use DB2 in a single AppServer

Note: If you set up the UDDI Registry application with a JDBC driver and datasource that reference Cloudscape, but set the *persist* property in *uddi.properties* to specify DB2, **or vice versa**, then some unexpected behavior will result, such as a fatal error on deleting an entity. If this happens, you should check that the above details are not in conflict. This only applies to a UDDI Registry installation on a single appserver.

7. Ensure that the UDDI Registry is configured appropriately for your installation, as described in the section on Configuring the UDDI Registry.
8. Stop and then restart the application server. On Unixz/OS platforms you must remember to run the *db2profile* script before issuing the *START appserver_proc_namestartServer.sh* command. This script is located within the DB2 instance's home directory under *SQLLIB* and can be invoked, for example, by typing:

```
". /home/db2inst1/sqllib/db2profile"
```

Continue with Configuring the UDDI Registry.

Setting up the UDDI Registry to use Cloudscape in a single appserver

Please see the section "Choice of database product to be used as the persistence store" to decide which database product you should use as your persistence store before proceeding further with this step.

If you plan to use Cloudscape for the database in which the UDDI Registry data will be held, use this task to setup and install the UDDI Registry application to use the supplied Cloudscape database.

This task configures Cloudscape on the host where you want to run the UDDI Registry. Before starting this task, ensure that Cloudscape is installed and running on that host, and that you can open a command window on that host.

This task, to configure Cloudscape for the UDDI Registry database. Cloudscape is supplied with WebSphere Application Server, so you should not need to install Cloudscape support.

This task, to configure Cloudscape for the UDDI Registry database, is part of the parent task to install and setup a UDDI Registry, described in Installing and

Setting up a UDDI Registry. You should complete this task at the appropriate step in the parent task, then return to and complete the parent task.

To configure Cloudscape for the UDDI Registry database, complete the following steps:

1. Copy the UDDI20 directory tree from the bin subdirectory of the deployment manager tree into the bin subdirectory of the target application server's install tree.
2. Copy the `uddicloudscapeuserfunc.jar` file from the lib subdirectory of the deployment manager install tree to the lib subdirectory of the target application server's install tree.
3. Ensure that the `persist` property in the `uddi.properties` file is set to `persist=Cloudscape`
4. Copy the `appserversetupuddi.jacl` script from the `UDDIReg/scripts` subdirectory of the deployment manager install tree to the WebSphere Application Server bin subdirectory (for example, on Windows, `C:\Progra~1\WebSphere\AppServer\bin`).
5. Change directory to the WebSphere Application Server bin subdirectory.
6. Start the application server on which the UDDI Registry is to run.
For example, you can start the application server `server1` by typing the command:

```
startserver server1
```

7. Create a JDBC driver and datasource to provide access to the UDDI20 Cloudscape database, and install the UDDI Registry application. To do this run the `WSADMIN` tool with the script `appserversetupuddi.jacl` as input, on the target application server, using the following command syntax:

(You should either run this script from the `UDDIReg/scripts` subdirectory where it is located, or copy it to some other suitable directory. Note that the `WSADMIN` command is located in the WebSphere bin subdirectory.)

```
wsadmin -f appserversetupuddi.jacl
        uddi-ear-location
        servername
        nodename
        WebSphere-lib-subdirectory
        cloudscapedbname
        > setupuddi.log
```

Where:

- `uddi-ear-location` is the fully-qualified path to the `uddi.ear` file in the `installableApps` subdirectory, specified using forward slashes regardless of platform. For example, on Windows:
`C:/Progra~1/WebSphere/AppServer/installableApps/uddi.ear`
- `servername` is the name of the application server on which the UDDI registry is to run; for example: `server1`. Note the the server name entered is case sensitive.
- `nodename` is the name of the WebSphere node on which the application server, `servername`, is running. Typically this is the machine name. Note that the node name entered is case sensitive.
- `WebSphere-lib-subdirectory` is the fully-qualified path to the WebSphere Application Server lib subdirectory, specified using forward slashes regardless of platform. For example:
 - On Windows: `C:/Progra~1/WebSphere/AppServer/lib`
 - On Unix: `/opt/WebSphere/AppServer/lib`

- *cloudscapedbname* is the fully-qualified path to the UDDI20 database within the bin subdirectory of your WebSphere AppServer installation, specified using forward slashes regardless of platform. For example:
 - On Windows, C:/Progra~1/WebSphere/AppServer/bin/UDDI20
 - On Unix, /opt/WebSphere/AppServer/bin/UDDI20
- > *setupuddi.log* is an optional parameter to direct the output to a log file as opposed to the default (which is to the screen)

The appserversetupuddi.jacl script will complete the following actions:

- a. Create a JDBC driver named UDDI.JDBC.Driver.<nodeName>.<server> and a datasource named UDDI.Datasource.<nodeName>.<server> (where <nodeName> is the name of the target node and <server> is the name of the target server), and will replace any existing driver and datasource of that name.
- b. Checks whether the WebSphere UDDI Registry application is already installed and, if so, stop the application and uninstall it.
- c. Installs the WebSphere UDDI Registry, then start it.

Note: The setup script, appserversetupuddi.jacl, cannot be used to install the UDDI Registry application into a clustered application server. It is possible to cluster the UDDI Registry application by installing UDDI into an unclustered application server using the setup script, and then clustering that application server.

Return to the next step in the parent task Installing the UDDI Registry into a single appserver.

Setting up the UDDI Registry to use DB2 in a single appserver

Please see the section "Choice of database product to be used as the persistence store" to decide which database product you should use as your persistence store before proceeding further with this step.

This task, to configure DB2 for the UDDI Registry database, is part of the parent task to install and setup a UDDI Registry, described in Installing and setting up a UDDI Registry. You should complete this task at the appropriate step in the parent task, then return to and complete the parent task.

If you plan to use DB2 for the database in which the UDDI Registry will be held, ensure that the correct prerequisite fix packs have been applied as listed at <http://www-3.ibm.com/software/webservers/appserv/doc/latest/prereq.html> otherwise the startup of the UDDI DB2 setup wizard will fail.

If you plan to use DB2 for the database in which the UDDI Registry data will be held, use this task to create and load the UDDI Registry database using DB2, and to setup and install the UDDI Registry application to use the database.

This task uses the UDDI DB2 setup wizard to configure DB2 on the system where you want to run the UDDI Registry. Before starting this task, ensure that DB2 is installed and running on that system.

Copy the UDDIReg directory tree from the deployment manager to the target application server where DB2 will run.

The following steps should be carried out on the system on which the target application server is located (referred to below as the 'target system').

1. On Windows, if you are using DB2 Version 7, then ensure that since installing DB2 you have run the usejdbc2.bat command file. For more information about this, see "Application Building Guide" in the DB2 documentation.
2. Create and load the UDDI Registry database, called UDDI20.

Note: If you are migrating from an earlier version of the UDDI Registry, and your UDDI20 DB2 database already exists, then you should skip this step unless you want to replace the existing database with a new UDDI20 DB2 database. If you do choose to replace an existing database then all of your existing UDDI data will be lost. **Important:** please also note that if you do choose to replace the existing UDDI20 database, then there must not be any applications or users accessing the database at the time that you run the UDDI DB2 setup wizard.

If you are planning to use a remote DB2 system on another host machine, you will need to copy the SetupDB2UDDI.jar file to the remote system and run it on that system to create and load the UDDI Registry database following the instructions within this bullet and continue with the next bullet (which states "If using a remote Db2 system on another host machine") on the local host.

The UDDI DB2 setup wizard used in this task will prompt you to provide the DB2 userid and password under which the UDDI Registry database will be created and subsequently accessed. Before starting this task, ensure that you have created an appropriate DB2 userid and password. This same userid and password must be used throughout the following steps where the DB2 userid and password is requested.

On Windows this should be a userid and password with administrative privileges. On Unix platforms, you should supply the userid and password of the DB2 instance in which you wish the database to be created.

5.0.1 + If you are using a non-English installation of DB2, then please ensure you have applied PTF1 before continuing.

5.0.1 + Information on how to do this and where to obtain the PTF can be found here.

5.0.1 + If you wish to implement the updated SetupDB2UDDI database wizard, then also please apply the PTF(s) mentioned above.

Note: **5.0.1 +** If you have a copy of the file SetupDB2UDDI.jar in your appserver directory, then the application of the base and Network Deployment PTFs will not update SetupDB2UDDI.jar in your appserver directory. You must apply the PTF for Network Deployment to your DeploymentManager file structure to update the SetupDB2UDDI.jar located there (in the /UDDIReg/scripts subdirectory), and then manually copy this jar to any appserver you may wish to run it on.

To create the database you use the UDDI DB2 setup wizard, which is supplied as a jar file called SetupDB2UDDI.jar in the UDDIReg/scripts subdirectoryshell script called SetupDB2UDDI.sh in the UDDIReg/bin subdirectory, by following these steps:

- a. Change directory to the directory containing the file SetupDB2UDDI.jar (that is, either the UDDIReg/scripts directory in which it is suppliedSetupDB2UDDI.sh (that is, either the UDDIReg/bin subdirectory, or a directory on the target system into which you have copied it).
- b. In order to run the wizard, you need to first ensure that you have access from your command line to the JVM supplied with WebSphere. This is done as follows:
 - On Windows, on a user with administrative privileges, in a command window type the following command:
`was_install\bin\setupcmdline.bat`

Where *was_install* is the path to the directory where you installed WebSphere Application Server.

For example, C:\Program Files\WebSphere\AppServer\bin\setupcmdline.bat

- On Unix platforms, at a command line, type one of the following commands:
 - If you are using bash, then as the root user run
`. /opt/WebSphere/AppServer/bin/setupCmdLine.sh`
 - If you are using csh, then as the root user run
`source /opt/WebSphere/AppServer/bin/setupCmdLine.sh`

where */opt/WebSphere/AppServer* is the path to the directory where you installed WebSphere Application Server.

- c. In the same command window, start the UDDI DB2 setup wizard by issuing one of the following commands:the following command:
`SetupDB2UDDI.sh`
 - To start a graphical user interface, type
`java -jar SetupDB2UDDI.jar`
 - To start a text mode interface, type:
`java -jar SetupDB2UDDI.jar -console`

Note: The above commands will fail unless your DB2 installation has the correct fix packs applied as listed on the prerequisite page found at <http://www-3.ibm.com/software/webservers/appserv/doc/latest/prereq.html>

- d. Follow the prompts to work through the wizard panels.
 - e. If necessary, check the log files for the wizard. A log file called UDDIloadDB.log is written out into the directory from which the wizard is run (but note that, on Windows platforms, if you have decided not to overwrite an existing UDDI20 database, then this fact will not be logged, and the log file will not be created).
3. If using a remote DB2 system on another host machine, refer to "Use of a remote DB2 database" and then return to this point and continue with the instructions below.
 4. Ensure that the persister property in the uddi.properties file is set to `persister=DB2`.
 5. On Unix, ensure that you have run the db2profile script to set up the environment for the DB2 instance that the UDDI Registry is using. For example, type the command:
`. /home/db2inst1/sqllib/db2profile`
 6. Start the application server on which the UDDI Registry is to run.

For example, you can start the application server `server1` by issuing the command:

```
startserver server1
START appserver_proc_name,JOBNAME=server1,
      ENV=cell_short_name.node_short_name.server1
```

Note: This command must be entered on a single line. It is split here for display purposes.

7. Copy the `appserversetupuddi.jacl` script from the `UDDIReg/scripts` subdirectory of the deployment manager install tree to the WebSphere Application Server `bin` subdirectory (for example, on Windows, `C:\Progra~1\WebSphere\AppServer\bin`).
8. Change directory to the WebSphere Application Server `bin` subdirectory.
9. Create a JDBC driver and datasource to provide access to the UDDI20 database, and install the UDDI Registry application. To do this run the `WSADMIN` tool with the script `appserversetupuddi.jacl` as input, on the target application server, using the following command syntax. (You should either run this script from the `UDDIReg/scripts` subdirectory where it is located, or copy it to some other suitable directory. Note that the `WSADMIN` command is located in the WebSphere `bin` subdirectory.)

```
wsadmin -f appserversetupuddi.jacl
      uddi-ear-location
      servername
      nodename
      WebSphere-lib-subdirectory
      dbname
      db2userid
      db2pwd
      db2-install-dir\location-of-db2java.zip
      > setupuddi.log
```

where

- `uddi-ear-location` is the fully-qualified path to the `uddi.ear` file in the `installableApps` subdirectory, specified **using forward slashes regardless of platform**.

For example, on Windows:

```
C:/Progra~1/WebSphere/AppServer/installableApps/uddi.ear
```

- `servername` is the name of the application server on which the UDDI registry is to run; for example: `server1`. Note that the name of the server is case sensitive.
- `nodename` is the name of the WebSphere node on which the application server, `servername`, is running. Note that the name of the node is case sensitive. Typically this will be the machine name.
- `WebSphere-lib-subdirectory` is the fully-qualified path to the WebSphere Application Server `lib` subdirectory, specified **using forward slashes regardless of platform**. For example:
 - On Windows: `C:/Progra~1/WebSphere/AppServer/lib`
 - On Unix or z/OS: `/opt/WebSphere/AppServer/lib`
- `dbname` is the name of the UDDI Registry database under DB2. You should specify `UDDI20` for this parameter

Note: If a remote DB2 system is being used the `dbname` stated here must be the alias created when access to the remote database was set up on the local system.

- `db2userid` and `db2pwd` are a valid DB2 userid and password with administrative privileges, as specified in an earlier step.

- *db2-install-dir* is the path under which you have installed DB2 on your system, specified **using forward slashes regardless of platform**.
 - For DB2 Version 7.2 (and later) on Windows the location is: C:/Progra~1/SQLLIB/java12/db2java.zip and note that the default value for *db2-install-dir* on Windows is C:\Program Files\SQLLIB.
 - For DB2 Version 8.1 (and later), then this parameter should be *db2-install-dir/java/db2java.zip*, and note that the default value for *db2-install-dir* on Windows is C:\Program Files\IBM\SQLLIB
- > *setupuddi.log* is an optional parameter to direct the output to a log file as opposed to the default (which is to the screen)

The appserversetupuddi.jacl will complete the following actions:

- a. Create a JDBC driver named UDDI.JDBC.Driver.<nodeName>.<server> and a datasource named UDDI.Datasource.<nodeName>.<server> (where <nodeName> is the name of the target node and <server> is the name of the target server), and will replace any existing driver and datasource of that name.
- b. Checks whether the WebSphere UDDI Registry application is already installed and, if so, stop the application and uninstall it.
- c. Installs the WebSphere UDDI Registry, then starts it.

Note: The setup script, appserversetupuddi.jacl, cannot be used to install the UDDI Registry application into a clustered application server. It is possible to cluster the UDDI Registry application by installing UDDI into an unclustered application server using the setup script, and then clustering that application server.

Return to the next step in the parent task Installing and Setting up a UDDI Registry.

Reinstalling the UDDI Registry application

If you wish to reinstall the UDDI Registry then please follow the appropriate section below.

Reinstalling into a deployment manager cell

If you wish to reinstall the UDDI Registry into the target application server, for example because you wish to alter certain aspects of its configuration using AAT, then you should rerun the setupuddi.jacl script (described in the appropriate link below):

- "Setting up the UDDI Registry to use Cloudscape within a deployment cell"
- "Setting up the UDDI Registry to use DB2 within a deployment cell"

Note: If you decide to change from using Cloudscape as your persistence store to DB2, or vice versa, then you should first remove UDDI from the application server using removeuddi.jacl. You can then run setupuddi.jacl to reinstall UDDI with the new type of persistence store.

Please also note that if you make such a change, then any data that you had previously stored will be lost.

Reinstalling into a single appserver

Remove the UDDI Registry application in the same manner as any other Enterprise Application and then install using the appropriate link shown below:

- Setting up the UDDI Registry to use Cloudscape in a single AppServer
- Setting up the UDDI Registry to use DB2 in a single AppServer

Note: If you decide to change from using Cloudscape as your persistence store to DB2, or vice versa, then you should first remove UDDI from the application server using `appserverremoveuddi.jacl`. You can then run `appserversetupuddi.jacl` to reinstall UDDI with the new type of persistence store.

Please also note that if you make such a change, then any data that you had previously stored will be lost.

Removing the UDDI Registry application from a deployment manager cell

If you wish to completely remove the UDDI Registry application from the target application server in the deployment manager cell, then you should run the `wsadmin` (`wsadmin.sh` on Unix Platforms) script `removeuddi.jacl`, which is located in the `UDDIReg/scripts` directory of the deployment manager install tree.

Please note that if the target server specified on invoking `removeuddi.jacl` is running at the same time, then the script will stop the server and will restart the server after when it has completed its operations.

The syntax for this script is:

```
wsadmin -f removeuddi.jacl
        servername
        nodename
        > removeuddi.log
```

Where `servername` and `nodename` are the server and node where you have deployed the UDDI Registry application. By default output will go to the screen, but, optionally, you can specify '`> removeuddi.log`' to direct output to a log file. For example,

```
wsadmin -f removeuddi.jacl server1 myriad
```

will remove the UDDI Registry application and related files from server `server1` running in node `myriad`, and will send any messages to the screen.

Removing the UDDI Registry application from a single appserver

If you wish to completely remove the UDDI Registry application from a stand-alone application server then you should run the `WSADMIN` script `appserverremoveuddi.jacl`, which will have been installed into the `UDDIReg/scripts` directory when you installed the UDDI Registry as part of a Network Deployment install.

Steps for this task are:

The syntax for this script is:

```
wsadmin -f appserverremoveuddi.jacl
        servername
        nodename
        > removeuddi.log
```

where

- *servername* and *nodename* are the name of the stand-alone application node in which it runs (these are the names that you specified when you ran `appserversetupuddi.jacl` to install the UDDI Registry application).
- by default output will go to the screen, but, optionally, you can specify '`> removeuddi.log`' to direct the output to a log file.

For example,

```
wsadmin -f appserverremoveuddi.jacl server1 monolith
```

will remove the UDDI Registry application and related files from server `server1` running in node `monolith`, and will send any messages to the screen.

Configuring the UDDI Registry

The UDDI Registry is supplied as a J2EE application file, `uddi.ear`. This is installed into the WebSphere Application Server during installation. If you want to change any of its configuration properties using AAT see "Configuring SOAP properties with the AAT".

If you enable WebSphere security then to run the publish API servlet of the IBM WebSphere UDDI Registry, you also need to configure WebSphere to use HTTPS and SSL, as described in Configuring WebSphere to use HTTPS and SSL

You can configure the following aspects of the UDDI Registry:

- Configuring global UDDI properties
- Modifying the database userid and password
- Configuring security properties
- Configuring the UDDI User Console (GUI) for multiple language encoding support
- Customizing the UDDI User Console (GUI)
- Configuring SOAP interface properties
- Configuring SOAP properties with the AAT
- Configuring SOAP properties in the deployment descriptor
- Configuring WebSphere to use HTTPS and SSL

Configuring global UDDI properties

To modify any of the global UDDI properties, you will need to edit the file called *uddi.properties*. More than one version of this file exists and the version you need to edit depends on

- whether you are in the installation phase or are updating the properties as a post installation step
- whether you are configured for a Deployment Manager or Base Application Server environment

The location of the file you should edit will be one of the following:

Deployment Manager Configurations

1. If you are in the process of installing the UDDI Registry application for the first time into a Deployment Manager cell and wish to make some generic changes before deploying it in the cell, the *uddi.properties* file will be located in the `<DeploymentManager-install-dir>/properties` directory. If you are reinstalling the UDDI Registry application into a Deployment Manager cell, then you should edit the file in the location described under 2 below.

Note: Please note that in a Deployment Manager configuration some properties (such as `persist` and `getServletURLprefix`) are dynamically set up in the `uddi.properties` file, during subsequent installation processing.

2. If the UDDI Registry is already configured into an application server within a Deployment Manager cell (i.e. you are undertaking post installation configuration changes), then the `uddi.properties` file you should edit will be located in the configuration repository, under the Deployment Manager filing system; that is in `<DeploymentManager-install-dir>/config/cells/<cellname>/nodes/<nodename>/servers/<servername>`, where `<cellname>` is the name of the Deployment Manager cell, `<nodename>` is the name of the node in which the application server is installed, and `<servername>` is the name of the application server in which you have installed the UDDI Registry.

Application Server Configurations

1. If you are in the process of installing the UDDI Registry application into an Application Server only environment you will be advised during the installation process when to make changes to the `uddi.properties` file.

Note: Please note that in contrast with the Deployment Manager configuration, UDDI properties are not dynamically set during installation processing.

2. If the UDDI Registry is already configured into a single application server which is **not** part of a Deployment Manager cell (i.e. you are undertaking post installation configuration changes), then the `uddi.properties` file will be located in the properties subdirectory of the WebSphere Application Server in which you have installed the UDDI Registry application, that is `<ApplicationServer-install-dir>/properties` directory.

The properties that can be changed within `uddi.properties` are as follows:

- The `dbMaxResultCount`, which is the limit on the number of rows of information that should be returned on Find requests, and will apply if the request does not specify a `maxRows` limit itself (or if it specifies a limit which exceeds this value). The initial value for this in `uddi.properties` is 100.
- The `persist`, which indicates what database is to be used as the persistence store for the UDDI Registry database. If you have installed the UDDI Registry into an application server within a Deployment Manager cell, then the `persist` property will have been set to the correct value for you. If you change this value, then you must also ensure that you have a UDDI Registry database created using the chosen database product (for more details about the UDDI Registry database, please refer to the section on "Installing the UDDI Registry"). You should also be aware that any data published to the UDDI Registry with one setting of the `persist` property will **not** be accessible when running the UDDI Registry application with a different setting for the `persist` property. The valid values for the `persist` property are:
 - `persist=DB2`
indicating that DB2 is to be used as the persistence store
 - `persist=Cloudscape`
indicating that Cloudscape is to be used as the persistence store

The initial value for this in `uddi.properties` is Cloudscape.

Note: This property is dynamically set by the `setupuddi.jacl` script when installing into a Deployment Manager cell so in this case you should not need to modify it.

- The default language to be used on a publish request as the `xml:lang` when one is not specified. The initial value for this in `uddi.properties` is `en-US`. This property must contain one of the valid `xml:lang` values.
- The UDDI site operator name. This is a string which is stored in every registry object, to indicate the operator of the UDDI Registry. The initial value for this in `uddi.properties` is `www.mycompany.com/uddi`. This property does not have any particular functional use, so its value can be set to any string that you feel is suitable.
- The maximum number of search keys that can be used on find API requests. The initial value for this in `uddi.properties` is 5.
- The `getServletURLprefix` and `getServletname` name, used to build up the discovery URL. The initial values for these are `http://localhost:9080/uddisoap/` and `get`. If you have installed the UDDI Registry into an application server within a Deployment Manager cell, then the `getServletURLPrefix` property will have been set for you using the value you specified as a parameter to the setup script. You are recommended to set suitable values for these properties before you first use the UDDI Registry.

Note: This property is dynamically set by the `setupuddi.jacl` script when installing into a Deployment Manager cell so in this case you should not need to modify it.

Applying these changes to your system

In order for your changes to take effect, you must do one of the following:

- If you are in the process of installing the UDDI Registry application for the first time, you should return to the original topic and complete the installation steps. Any changes you have made will be picked up during this subsequent processing.
- If you have made post installation changes in a base Application Server only environment, you should stop and restart the UDDI Registry application using the WebSphere Administrative Console.
- If you have made post installation changes in a Deployment Manager environment you should:
 1. Execute a Full Resynchronization for the node where the UDDI Registry executes. This can be done from the WebSphere Network Deployment Administrative Console under section Systems Administration ==> Nodes. Select your node, and then click the button for 'Full Resynchronization'. It is important that you do a 'Full Resynchronization' and not just a 'Synchronize'.
 2. Stop and restart the UDDI Registry application using the WebSphere Administrative Console

Modifying the database userid and password

If you use DB2 as the persistence store for the UDDI Registry, and you need to change the database userid and/or password, you should alter the user and password values in the custom properties of the 'UDDI Datasource', which can be edited from the WebSphere Administrative Console. The UDDI.Datasource is under datasources within the UDDI.JDBC.Driver, which is itself found under JDBC Providers under Resources. Do not alter the `databaseName`.

Configuring security properties

See *Configuring WebSphere to use HTTPS and SSL* for details on configuring security properties.

To run the publish API servlet of the IBM WebSphere UDDI Registry, you also need to configure WebSphere to use HTTPS and SSL, as described in *Configuring WebSphere to use HTTPS and SSL*.

Configuring the UDDI User Console (GUI) for multiple language encoding support

If you want to use multiple language encoding support in the User Console (GUI), you need to configure the application server into which the UDDI Registry application is installed with UTF-8 encoding enabled. To do this, please refer to "Configuring application servers for UTF-8 encoding" elsewhere in the WebSphere Application Server version 5 InfoCenter on enabling an application server for UTF-8.

Customizing the UDDI User Console (GUI)

The look and feel of the UDDI Console is determined by the styles defined in the *uddi_gui.css* file which is located in the `/gui.war/theme` directory of the installed UDDI Registry application directory. The UDDI Registry application directory will be one of the following, depending on where you have installed the UDDI Registry:

- If you have installed the UDDI Registry into an application server within a Deployment Manager cell, the directory is `UDDIReg.ear` under the `/apps` directory of the configuration repository for the cell.
- If you have installed the UDDI Registry into a single application server which is not part of a Deployment Manager cell, the directory is `UDDIRgistry.ear` under the `installedApps` directory of the WebSphere Application Server in which you have installed the UDDI registry application.

The contents of this file can be edited to change the colors, fonts and font sizes according to the user's preference.

The content and layout of the UDDI User Console is provided by JSP pages, which can be customized by a programmer who is familiar with JSPs. The JSP pages are found in the `uddi.ear` enterprise application, which is under the `installedApps` subdirectory of the WebSphere AppServer installation. To locate the JSPs, you should expand the `UDDI_Registry.ear`, open the `gui.war`, and they are located under `WEB-INF` in the `pages` subdirectory. So, on a Windows system which has WebSphere installed in the default location, the JSP files will be found in "`<WebSphere-install-dir>\installedApps\UDDI_Registry.ear\gui.war\WEB-INF\pages`". These JSP pages also contain some application logic (as opposed to presentation logic) which should not be changed.

Configuring SOAP interface properties

You can configure the following SOAP interface properties

- *defaultPoolSize* - the number of SOAP parsers with which to initialize the parser pool for the SOAP interface. The can be set independently for the Publish (`uddipublish`) and Inquiry (`uddi`) APIs. For example, if you expect more

inquiries than publish requests via the SOAP interface, you can set a larger pool size for the Inquiry API. The default initial size for both APIs is 10.

- The *context root* used for the Publish and Inquiry APIs, which forms a part of the URL by which they are accessed. By default this is /uddisoap.
- Whether the API is to be secure (via HTTPS) or insecure (via HTTP). The default is to use HTTPS.

To configure the following SOAP interface properties, you can use either of the following methods, as described below:

- Configuring SOAP properties with the AAT (the recommended option, especially for a production environment)
- Configuring SOAP properties in the deployment descriptor for the SOAP module in the UDDI application directly. This option is faster and may be the preferred method in a test environment.

Configuring SOAP properties with the AAT

To configure SOAP properties by using the WebSphere Application Assembly tool, complete the following steps:

- Select *Update* and click on the Application icon.
- Select the *uddi.ear* file (this is placed, by the UDDI installation, into the UDDI install directory (e.g. C:\WebSphere\installableApps\uddi.ear).
- Expand the *uddi.ear* icon on the left-hand pane in the AAT.
- Expand the *Web Modules* tree.
- Expand the *uddi Soap* tree
- To change the *defaultPoolSize*, expand Web Components and then *uddipublish* (for the publish API) or *uddi* (for the inquiry API).
 - Click on *Initialization Parameters* which will show the *defaultPoolSize* parameter in the upper right-hand pane. This can be edited in the lower right-hand pane.
- To change the *context root*, click on *UDDI Soap* which will display general information about the SOAP module in the lower right-hand pane in AAT. The *context root* can be edited in this pane.
- To change the publish API to use HTTP (instead of HTTPS), click on *Security Constraints* and change the *Transport Guarantee* from Confidential to none.
- Having made any changes above, you must now save them. To do this, click on *File -> Save (or Save As)* to save your changes.
- Redeploy the *uddi.ear* to WebSphere, by first removing it and reinstalling it via the Administrator's Console.

Configuring SOAP properties in an already-deployed application

To configure SOAP properties after the UDDI application has been installed, complete the following steps:

1. The deployment descriptor for the SOAP module (*web.xml*) is found in the *WEB-INF* subdirectory of the *uddi.ear* application in the installed applications within the WebSphere install directory (for example, <WebSphere-install-dir>\installableApps\uddi.ear\soap.war\WEB-INF). It can be edited directly to specify the desired settings.
2. Stop and restart the WebSphere Application server for the changes to take effect.

Configuring WebSphere to use HTTPS and SSL

To support the use of secure access with the IBM WebSphere UDDI Registry, you need to configure WebSphere to use HTTPS and SSL. Please refer to the information available elsewhere in this InfoCenter for configuring SSL in WebSphere Application Server. It is assumed throughout the information for the UDDI Registry that, where SSL is used, it has been configured on port 9443.

Administering the UDDI Registry

This section describes the various tasks about administering the UDDI Registry.

- Running the UDDI Registry
- Backing up and restoring the UDDI Registry database

Running the UDDI Registry

The UDDI Registry is started automatically when the Application Server is started. In order to stop and restart it, use the Administrative console.

Backing up and restoring the UDDI Registry database

If you want to protect the data in your UDDI Registry database, you can backup and restore the database using the facilities of the database product. For DB2, you can do this by using the export and import functions of the DB2 Control Center. For Cloudscape you can simply use operating system tools to copy the database directory. Please refer to the database product information for more details.

The UDDI Registry database is called UDDI20, and the tables which should be backed up are:

- ADDRESS
- ADDRLINE
- BSERVICE
- BTEMPLATE
- BUSINESS
- CATEGORY
- CATEGORYBAG
- CONTACT
- DESCR
- DISCOVERYURL
- EMAIL
- EXTCATEGORY
- IDENTIFIERBAG
- INSTANCEDETAIL
- NAMEELEMENT
- OVERVIEWDOC
- PHONE
- PUBLISHERASSERTION
- SERVICEPROJECTION
- TMODEL
- VALIDATIONCACHE
- VALIDATIONSERVICES

UDDI user console

This topic describes the layout of the UDDI user console, also referred to as the Graphical User Interface (GUI), which you can use to interact with the IBM WebSphere UDDI Registry.

For information about how to display the UDDI user console, see [Displaying the user console](#).

If you will be using the UDDI Console, then it is recommended that you configure the application server into which you have installed the UDDI Registry for UTF-8 encoding support: please refer to the section on "Configuring the UDDI User Console for multiple language encoding support".

- The user console provides a graphical user interface to the majority of the UDDI Version 2 API. It is not intended to support the full API set: there is some focus on inquiry operations, as the main purpose of the UDDI User Console is to allow users to issue inquiry requests and to familiarize themselves with general UDDI concepts. This section documents those areas for which support through the user console is not provided, together with other known restrictions to the user console.
 - General
 - Help is provided in the form of explanatory text on the screens.
 - Maximum rows cannot be specified on finds. The single maximum rows value for the registry can be set through the *dbMaxResultCount* global configuration property. For more information on setting this property see [Configuring global UDDI properties](#)
 - Find business
 - The business identifier feature is not supported.
 - Find service type
 - The business identifier feature is not supported.
 - Add business
 - The business contact must be supplied as a name and role (no other information is supported).
 - Add service type
 - The overview URL can be entered, but only with one description in English.
 - Add service
 - There is no support for entering a Hosting Redirector, nor for adding an overviewURL.
- **Note:** The UDDI Version 2 specification states that when a tModel is deleted, it should not be physically deleted. This allows the tModel to be reinstated. One effect of this is that, if you delete a tModel using the UDDI user console, the tModel is still visible through the Show Owned Entities display.

The UDDI user console is split into three distinct areas. At the top of the screen are buttons which activate various functions in the areas below this bar. These buttons are:

Home returns you to the IBM WebSphere UDDI Registry welcome page

Find activates the Find tab on the frame below to the left

Publish

similarly activates the Publish tab on the frame below to the left

Below the WebSphere UDDI Registry banner the screen is split into two parts. On the left are the two tabs mentioned above, the Find and Publish tabs.

Find tab

The Find tab is in two parts. At the top, a **Quick Find** service is provided. There are three radio buttons to enable a choice of 'service', 'business' and 'technical model' finds. Below these radio buttons is a text entry box for entering the name to search for and, beneath this, a 'Find' link to start the search. Comments are provided to show the user the wildcard character. The results of clicking on the 'Find' link are shown in the detail frame to the right.

Beneath the Quick Find is a section for **Advanced Find** functions which enables the user to choose which entity they want to perform an advanced search on. There are three links: Find services, Find businesses and Find technical models. Clicking one of these links displays the corresponding advanced search form in the frame to the right, which the user may use to enter search criteria. To initiate a Find, the user must first enter a search path (the % wildcard may be used) and then click the blue Add link to enter the search. Then click on the 'Find Services' (or 'Find Businesses/Find technical models) link below to initiate the Find operation. The **Locator** section has a link (marked in blue with the words "**Show category tree**") which displays the tree from which the user can select categories (or taxonomies). This is shown in the left-hand frame. In the advanced search form there are two links to start the search (mid-way down and at the bottom).

The results of clicking either of the two links to start the search are displayed in the same detail frame.

Publish tab

The *Publish* link on the top banner activates the Publish tab in the navigation frame to the left. The Publish tab is split into three distinct sections.

1. Quick Publish Function

The top part is a **Quick Publish** section to allow the user to publish a business or technical model by name only. There are two radio buttons to enable a choice of 'business' or 'technical model'. Below these radio buttons is a text entry box for entering the name to assign to the selected entity and, beneath this, a blue 'Publish now' link to publish the entity. The results of clicking on the **Publish now** link are shown in the detail frame to the right.

2. Advanced Publish Functions

To publish an entity with more detail, such as with multiple names, descriptions and categories, use the **Advanced Publish** section below this. The comments below each link ('Add a business' and 'Add a technical model') describe individual functions. Clicking one of these links displays the corresponding advanced publish form in the detail frame where the user may enter details about the entity they want to publish. As in the Advanced Find functions described above, there are two links to publish a business or technical model (one towards the top of the form and the other at the bottom). Similarly the **Locator** section allows taxonomies to be shown in the left frame from which the user can select categories.

Following entry of the relevant details on the **Advanced Publish** section, the user must click on the **Publish Business** bar in order for the business to be published to the UDDI Registry.

3. Registered Information

Below the Advanced Publish section is a **Registered Information** section which has a link to **Show Owned Entities** in order to show the businesses, services and technical models registered to the individual user, and pending business relationships. Clicking the **Show Owned Entities** link displays the **Show**

Owned Entities page in the detail frame at the right. The **Show Owned Entities** page is organized in three sections: **Registered Businesses**, **Pending Business Relationships** and **Registered Technical Models**. Each section shows the number of registered items.

Edit and Delete Businesses

Users can **Edit** or **Delete** businesses owned by them by clicking the appropriate links in the **Actions** column.

After an **Edit** or **Delete** function has been completed, the user **must** click on the **Update Business** bar in order to publish the changes to the UDDI Registry.

After Deleting a Business the user must confirm the deletion by clicking on the 'Delete this Business' link.

Adding a Service to a Business

Services are added to a business by clicking the **Add a Service** link in the **Services** column of the **Registered Businesses** section.

After the **Add a Service** function is complete, users **must** click on the **Publish Service** bar in order to publish the service to the UDDI Registry.

Referencing a Service from a Business

Services can also be 'referenced' by a business as if the business was the owner of the service. This 'service projection' is performed by clicking the **Reference a service** link in the **Services** column. Services associated with a business, whether they are owned or referenced, can be displayed by clicking the **Show services** link. This acts as a toggle between displaying services available for editing or deleting, and hiding them.

Adding a relationship to another Business

A business can be associated with another business in the UDDI Registry and this function is performed by clicking the **Add a relationship** link in the **Actions** column of the Registered Businesses section. Clicking the **Show related businesses** link in the **Actions** column displays a list of any completed business relationships.

The **Pending Business Relationships** section shows all incomplete publisher assertions, where only one party has asserted a relationship and is waiting for the other party to make the same assertion. This section reminds the user of any relationships that involve their businesses. Once both parties have asserted the same relationship between two businesses, the relationship moves from the **Pending Business Relationships** section and appears in the list of relationships displayed after clicking the **Show related businesses** link in the **Registered Businesses** section.

Technical Models

Technical Models owned by the user are shown in the bottom **Registered Technical Models** section. As for businesses, users can Edit or Delete technical models owned by them by clicking the appropriate links in the **Actions** column.

Note: Users should take note that deletion of Technical Models (tModels) does **not** cause them to be physically deleted, but hidden. This is in accordance with the UDDI Registry V2.0 specifications. After deletion Technical Models are shown under the "**Shown Owned Entities**" link on the publish page but not via the Find links on the Find page. ALL other entities are deleted from the UDDI Registry in the normal way.

Example of publishing a Business, Service and tModel with the User Console

For the example, here, we will assume a business called Mondeo Cars that sells used cars

1. Add the Business

Click on the Publish tab in the left hand navigation frame. Then click on 'Add a business' in the Advanced Publish in the left pane. This takes you to a 'Publish Business' pane on the right. Start by adding your Business Name in the text field labelled (Mondeo Cars in this example) and select a language and then click on the blue Add link to the right. This adds the business name (but the business is not yet published - more about which is explained later). Below the Business Name is an area called Descriptions - it allows free text to be added to describe the business - if you enter anything here you must click on the blue Add link to the right to insert the description.

The next section/area is the Locator area which can be used to describe the business according to what categories it falls into. This example uses a Used car dealership. Within the NAICS taxonomy (which you may view by clicking on 'Show category tree' and then expanding NAICS) this is a Retail Trade [44] entry which, on expansion, has Motor Vehicle and Parts Dealers [441] and, again on expansion, has automobile Dealers [4411] and Used Car Dealers [44112]. This fits the Business perfectly, so clicking on Used Car Dealers will enter the Key Name and Key Value into the business. For Checked Categorizations (such as NAICS) the Key name is not checked but the Key Value is checked. It should be noticed that for unchecked categorizations (such as 'other' or unspc') the Key value is not checked either. If the locator field has been added, then the blue Add link must be clicked. The final area is Contacts, which can have names and role information added if required. Again, the blue Add link must be clicked after adding the relevant information.

Once all the fields are filled in to the required level, the final action is to click on the Publish Business at the bottom of the form or at the top. This causes the business to be published to the UDDI Registry and a page is displayed showing the business details.

2. Add a Service

From the Publish tab, there is a 'Show owned entities' link. This shows the businesses owned by the current user in the Registry and the language to be used for a particular user. For Mondeo Cars, the user will see a 'Add service' button. Clicking this button shows the Publish Service form. The top part of the form is the Service Name field. After adding this name, the user must click on the blue Add link to enter the name. As in the Adding the Business form, each subsequent part must end with the blue Add link being clicked to add that part of the information to the service. The sections are (from top to bottom, Description (a free text area), Access Points (to add link points to the Service), Locator (to add references to taxonomies to the service), and Technical Models (to associate existing tModels to the Service). After completion of those areas required, clicking on the 'Publish Service' button will Publish the service to the UDDI Registry with the current form contents.

3. Adding a new technical model

Clicking on the 'Add a technical model' link in the left frame opens up the Publish Technical Model form on the right. A tModel can only have one name - hence the lack of a blue Add link next to the Technical Model Name field. Beneath this field are other fields - Description (a free text area to describe the technical model), Locator (to describe the technical model with taxonomies, and an Overview URL (which gives a URL pointing to an overview document, a description of the document and a Language field). For each of these fields there is a blue Add link which must be clicked to add the relevant data. At the bottom of the form is a 'Publish Technical Model' link which will create the technical model in the UDDI Registry.

There is a Publish link at the top of the frame in each case also - after the Name section.

- Displaying the user console

Displaying the user console

This topic describes how to display the UDDI Registry user console (sometimes referred to as the GUI). The URL you use depends on whether or not you have enabled WebSphere security:

- If you have the WebSphere security disabled, you can access the UDDI User Console by using the following URL in your Web browser:

`http://<hostname>:9080/uddigui`

Note: With WebSphere security disabled, all the publish operations are performed using a userid of `UNAUTHENTICATED`.

- If you have WebSphere security enabled, you can access the UDDI User Console through HTTPS by using the following URL in your Web browser:

`https://<hostname>:9443/uddigui`

The User Console displays the default frameset containing the header frame, navigation frame showing find options, and details frame. When you click the link to show the publish options in the navigation frame, you are challenged for a userid and password.

If WebSphere security is enabled and you try to access a publish action via an unsecured link, e.g. clicking the publish link on the navigation frame where the User Console was opened with

`http://<hostname>:9080/uddigui`

you will be redirected to a secure logon. Inquire functions will work as expected.

SOAP Application Programming Interface for the UDDI Registry

Access to the SOAP API will by default be available at:

`http://localhost:9080/uddisoap/inquiryapi`

or

`https://localhost:9443/uddisoap/publishapi`

Where 'localhost' is the address by which your WebSphere server is known. If security is enabled on your WebSphere server, the `publishapi` will also be protected by basic-authentication. By default, when security is enabled, the `publishapi` is restricted to HTTPS, this is to ensure the confidentiality and security of your data whilst in transit to UDDI. If you do not wish to use SSL, when security is enabled, you will have to modify the jar file using AAT to remove the `CONFIDENTIAL` restriction placed upon the publish URLs. For more information about this topic, see the section on **Configuring SOAP properties with the AAT**. If you normally access your WebSphere server via a web server, you will need to ensure the plugin

configuration for the WebSphere plugin on the web server has been updated since installing UDDI. This will then allow access to the UDDI SOAP API through the URLs

<http://localhost/uddisoap/inquiryapi>

or

<https://localhost/uddisoap/publishapi>

Where 'localhost' is the address by which your web server is accessed. Note that if you plan on accessing UDDI via a web server in this manner, that the samples will require modification to inform them of the SSL certificates used by your web server, so that the samples can make SSL connections to the web server. It is beyond the scope of this document to cover the many variants available on web server/WebSphere/java SSL configurations

- Using the SOAP API
- Handling Errors as a User of the SOAP API

Programming the UDDI SOAP API

To use the SOAP API construct a properly formed UDDI message within the body of a SOAP request, and send it using HTTP POST to the URL of the API which the request relates to. The response will be returned within the body of the HTTP reply. Although the samples are written in Java, you may use other programming languages to create your SOAP client, providing you still send requests compliant to the SOAP specification. Valid UDDI requests should conform to the UDDI schema, and be as detailed within the UDDI standard documentation available from:

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

For more information on using the SOAP API, refer to "The UDDI Registry application programming interface" section within this InfoCenter.

SOAP API error handling tips in the UDDI Registry

When using the SOAP API there are three main categories that may cause an error to be returned:-

- An invalid/incorrect request being sent to the SOAP API. e.g. Incorrectly formed XML, Badly formed UDDI requests, Non-schema compliant requests.
- Invalid business logic within a SOAP API request. e.g. Attempting to delete a business that does not exist.
- Problems occurring while processing a valid request. e.g. Server connection to database failure.

In each of these cases, an error will be returned to the client that made the request, which will attempt to explain further what the problem was.

UDDI Registry Application Programming Interface

The IBM WebSphere UDDI Registry fully supports the Application Programming Interface (API) specification which can be viewed by visiting http://www.uddi.org/pubs/ProgrammersAPI_v2.pdf. Any changes from this specification are documented within the IBM WebSphere UDDI Registry information.

- The Inquiry API
- The Publish API

Inquiry API for the UDDI Registry

The Inquiry API provides three forms of query that follow broadly used conventions which match the needs of software traditionally used within registries.

- The browse pattern
- The drill-down pattern
- The invocation pattern
- Inquiry API functions

Browse pattern for the UDDI Registry

Software that allows people to explore and examine data - especially hierarchical data - requires browse capabilities. The browse pattern characteristically involves starting with some broad information, performing a search, finding general result sets and then selecting more specific information for drill-down.

The UDDI API specifications accommodate the browse pattern by way of the *find_xx* API calls. These calls form the search capabilities provided by the API and are matched with summary return messages that return overview information about the registered information that is associated with the inquiry message type and the search criteria specified in the inquiry.

A typical browse sequence might involve finding whether a particular business you know about has any information registered. This sequence would start with a call to *find_business*, perhaps passing the first few characters of a business name that you already know. This returns a *businessList* result. This result is overview information (keys, names and descriptions) derived from the registered *businessEntity* information, matching on the name fragment that you provided. If you spot the business you are looking for within this list, you can drill down into the corresponding *businessService* information, looking for particular technical models (e.g. purchasing, shipping, etc) using the *find_service* API call. Similarly, if you know the technical *fingerprint* (tModel signature) of a particular software interface and want to see if the business you've chosen provides a web service that supports that interface, you can use the *find_binding* inquiry message.

Drilldown pattern for the UDDI Registry

When you have a key for one of the four main data types managed by a UDDI registry, you can use that key to access the full registered details for a specific data instance. The UDDI data types are *businessEntity*, *businessService*, *bindingTemplate* and *tModel*. You can access the full registered information for any of these structures by passing a relevant key type to one of the *get_xx* API calls.

Continuing the example from the previous section on browsing, one of the data items returned by all of the *find_x* return sets is key information. In the case of the

business we were interested in, the `businessKey` value returned within the contents of a `businessList` structure can be passed as an argument to `get_businessDetail`. The successful return to this message is a `businessDetail` message containing the full registered information for the entity whose key value was passed. This will be a full `businessEntity` structure.

Invocation pattern for the UDDI Registry

In order to prepare an application to take advantage of a remote Web service that is registered within the UDDI registry by other businesses or entities, you need to prepare that application to use the information found in the registry for the specific service being invoked.

The `bindingTemplate` data obtained from the UDDI registry represents the specific details about an instance of a given interface type, including the location at which a program starts interacting with the service. The calling application or program should cache this information and use it to contact the service at the registered address whenever the calling application needs to communicate with the service instance. In previously popular remote procedure technologies tools have automated the tasks associated with caching (or hard coding) location information. Problems arise however when a remote service is moved without any knowledge on the part of the callers. Moves occur for a variety of reasons, including server upgrades, disaster recovery, and service acquisition and business name changes.

When a call fails using cached information previously obtained from a UDDI Registry, the proper behavior is to query the UDDI Registry for fresh `bindingTemplate` information. The proper call `get_bindingDetail` passing the original `bindingKey` value. If the data returned is different from the cached information, the service invocation should automatically retry the invocation using the fresh information. If the result of this retry is successful, the new information should replace the cached information.

By using this pattern with Web services, a business using a UDDI Registry can automate the recovery of a large number of partners without undue communication and coordination costs. For example, if a business has activated a disaster recovery site, most of the calls from partners will fail when they try to invoke services at the failed site. By updating the UDDI information with the new address for the service, partners who use the invocation pattern will automatically locate the new service information and recover without further administrative action.

Inquiry API functions in the UDDI Registry

These messages represent inquiries that can be made of the UDDI Registry. These messages all behave synchronously.

The queries available are:

find_binding

Used to locate specific bindings within a registered `businessService`. Returns a `bindingDetail` message.

find_business

Used to locate information about one or more businesses. Returns a `businessList` message.

find_relatedBusinesses

Used to locate information about `businessEntity` registrations that are related to a specific business entity whose key is passed in the inquiry. The

Related Businesses feature is used to manage registration of business units and subsequently relate them based on organizational hierarchies or business partner relationships. Returns a relatedBusinessList message.

find_service

Used to locate specific services within a registered businessEntity. Returns a serviceList message.

find_tModel

Used to locate one or more tModel information structures. Returns a tModelList structure.

get_bindingDetail

Used to get full bindingTemplate information suitable for making one or more service requests. Returns a bindingDetail message.

get_businessDetail

Used to get the full businessEntity information for one or more businesses or organizations. Returns a businessDetail message.

get_serviceDetail

Used to get full details for a given set of registered businessService data. Returns a serviceDetail message.

get_tModelDetail

Used to get full details for a given set of registered tModel data. Returns a tModelDetail message.

Accessible query values in the UDDI Registry:

A list of the accessible queries for the UDDI Registry is given here.

Accessible queries within the UDDI Registry are:

find_binding

used to locate specific bindings within a registered businessService. Returns a bindingDetail message that contains zero or more bindingTemplate structures matching the criteria specified in the argument list.

find_business

used to locate information about one or more businesses. Returns a businessList message that matches the conditions specified in the arguments.

find_relatedBusinesses

used to locate information about businessEntity registrations that are related to a specific business entity whose key is passed in the inquiry. The Related Businesses feature is used to manage registration of business units and subsequently relate them based on organizational hierarchies or business partner relationships. Returns a relatedBusinessList message containing results that match the conditions specified in the arguments.

find_Service

used to locate specific services within a registered businessEntity. Returns a serviceList message that matches the conditions specified in the arguments.

find_tModel

used to locate a list of tModels that match a set of specified criteria. The response will be a list of abbreviated information about registered tModel data that matches the criteria specified. The result will be returned in a tModelList message.

get_bindingDetail

used to requesting the run-time bindingTemplate information for the purpose of invoking a registered business API. Returns a bindingDetail message.

get_businessDetail

used to return complete businessEntity information for one or more specified businessEntity registrations matching on the businessKey values specified.

get_businessDetailExt

used to return extended businessEntity information for one or more specified businessEntity registrations. This message returns exactly the same information as the get_businessDetail message, but may contain additional attributes if the source is an external registry that is compatible with the API specification.

get_serviceDetail

used to request full information about a known businessService structure. Returns a serviceDetail message.

get_tModelDetail

get_tModelDetail

For full details of the syntax of the above queries, please refer to the API Specification at <http://www.uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.pdf>

Publish API for the UDDI Registry

The messages in this section represent commands that are used to publish and update information contained in a UDDI registry. The messages defined in this section all behave synchronously.

The Publishing API calls defined that UDDI operators support are:

add_publisherAssertions

this call causes one or more publisherAssertion to be added to an individual publisher's assertion collection.

delete_binding

causes one or more instances of bindingTemplate data to be deleted from the UDDI registry.

delete_business

used to remove one or more business registrations and all direct contents from a UDDI registry.

delete_publisherAssertions

causes one or more publisherAssertion elements to be removed from a publisher's assertion collection.

delete_service

is used to remove one or more businessService elements from the UDDI registry and from its containing businessEntity parent.

delete_tModel

is used to logically delete one or more tModel structures. Logical deletion hides the deleted tModels from find_tModel result sets but does not physically delete it, so it is returned on a get_registeredInfo request.

discard_authToken

is used to inform an Operator Site that the authentication token is to be discarded, effectively ending the session. Subsequent calls that use the same authToken will be rejected. This message is optional for Operator Sites that do not manage session state or that do not support the get_authToken message.

get_assertionStatusReport

this call provides administrative support for determining the status of current and outstanding publisher assertions that involve any of the business registrations managed by the individual publisher account. Using

this message, a publisher can see the status of assertions that they have made, as well as see assertions that others have made that involve businessEntity structures controlled by the calling publisher account.

get_authToken

the call used to obtain an authentication token. Authentication tokens are opaque values that are required for all other publisher API calls. This message is not required for Operator Sites that have an external mechanism defined for users to get an authentication token. This API is provided for implementations that do not have some other method of obtaining an authentication token or certificate, or that choose to use user ID and password based authentication.

get_publisherAssertions

this is used to obtain the full set of publisher assertions that are associated with an individual publisher account. Publisher assertions are used to control publicly visible business relationships.

get_registeredInfo

this call is used to get an abbreviated list of all businessEntity and tModel data that are controlled by the individual associated with the credentials passed.

save_binding

is used to save or update a complete bindingTemplate element. this message can be used to add or update one or more bindingTemplate elements as well as the container/contained relationship that each bindingTemplate has with one or more existing businessService elements.

save_business

this is used to save or update information about a complete businessEntity element. This API has the broadest scope of all the save_xx API calls in the publisher API, and can be used to make sweeping changes to the published information for one or more businessEntity elements controlled by an individual.

save_service

the call used to add or update one or more businessService elements exposed by a specified businessEntity.

save_tModel

this call adds or updates one or more registered tModel elements.

set_publisherAssertions

this call is used to manage all of the tracked relationship assertions associated with an individual publisher account.

For full details of the syntax of the above queries, please refer to the API Specification at http://www.uddi.org/pubs/ProgrammersAPI_v2.pdf.

UDDI EJB Interface for the UDDI Registry

This section describes how to use the EJB application programming interface (API) of the IBM WebSphere UDDI Registry component to publish, find and delete UDDI entries.

The necessary client classes are contained in the *uddiejbclient.jar* file in the *ejb* subdirectory of the *UDDIReg* directory under the WebSphere appserver directory tree.

The javadoc for the EJB API is contained in the javadoc directory tree under the *ejb* subdirectory of the *UDDIReg* directory under the WebSphere appserver directory tree.

The EJB API is contained in two stateless session beans, one for the Inquiry API (`com.ibm.uddi.ejb.InquiryBean`) and one for the Publish API (`com.ibm.uddi.ejb.PublishBean`), whose public methods form an EJB interface for the UDDI Registry. All the public methods on the `InquiryBean` correspond to UDDI Inquiry API functions, and all the public methods on the `PublishBean` correspond to UDDI Publish API functions. (Not all UDDI API functions are implemented, e.g. `get_authToken`, `discard_authToken`, `get_businessDetailExt`, etc.) For version 1 of the UDDI registry, the EJB component supports only UDDI v2.0.

The two EJBs use container-managed transactions. The transaction attribute for the methods of the `InquiryBean` is `NotSupported`, and for the methods of the `PublishBean` it is `Required`. You should not change the transaction attributes as this could result in undesirable behavior.

Within each interface there are groups of overloaded methods that correspond to the operations in the UDDI 2.0 specification. There is a separate method for each major variation in function. For example, the single UDDI 2.0 operation `find_business` is represented by 10 variations of `findBusiness` methods, with different variations for finding by name, finding by categoryBag etc.

The arguments for the EJB interface methods are java objects in the package `com.ibm.uddi.datatypes`. Roughly speaking, there is a one-one correspondence between classes in this package and elements of the UDDI v2.0 XML schema. Exceptions to this are, for example, where UDDI XML elements can be represented by a single String. (See Package `com.ibm.uddi.datatypes` below for more information.)

Enabling an EJB Client

This section is written on the assumption that WebSphere Application Server V5.0, a supported database and the IBM WebSphere UDDI Registry have already been installed.

Classpaths

In order for EJB clients to work correctly, the following jar files and folders must be added to the user's CLASSPATH:

For Windows

<code><WebSphere-install-dir>\lib\j2ee.jar</code>
<code><WebSphere-install-dir>\lib\naming.jar</code>
<code><WebSphere-install-dir>\lib\namingclient.jar</code>
<code><WebSphere-install-dir>\lib\ecutils.jar</code>
<code><WebSphere-install-dir>\lib\sas.jar</code>
<code><WebSphere-install-dir>\properties</code>

For Unix Platforms, including z/OS

<code><WebSphere-install-dir>/lib/j2ee.jar</code>
<code><WebSphere-install-dir>/lib/naming.jar</code>
<code><WebSphere-install-dir>/lib/namingclient.jar</code>

<WebSphere-install-dir>/lib/ecutils.jar
<WebSphere-install-dir>/lib/sas.jar
<WebSphere-install-dir>/properties

In addition to these jars, there is also the jar file that contains all of the UDDI specific API for the EJB interface, which can be found at:

For Windows

<DeploymentManager-install-dir>\UDDIReg\ejb\uddiejbclient.jar

where <DeploymentManager-install-dir> is the install location for WebSphere Application Server for Network Deployment, which by default is C:\Progra~1\WebSphere\DeploymentManager.

For Unix Platforms, including z/OS

<DeploymentManager-install-dir>/UDDIReg/ejb/uddiejbclient.jar

where <DeploymentManager-install-dir> is the install location for WebSphere Application Server for Network Deployment, which by default is /opt/WebSphere/DeploymentManager for Linux/Solaris systems or /usr/WebSphere/DeploymentManager for AIX systems.

The Path

Please ensure that your PATH statement starts with <WebSphere-install-dir>\java\bin

Creating an EJB Client

If you want to read about creating EJB Clients in more detail, then please read the "Sun Microsystems Enterprise JavaBeans™ Specification Version 2.0"

Finding the EJB Reference

An EJB Client can be a stand-alone Java application, an Applet, Servlet or a JSP. This document only covers writing a stand-alone Java application. In order to invoke an EJB that has been deployed into WebSphere on the server side, the Client must do two things: find the EJB on the server, and then create a Client side reference to that EJB. Once this Client side reference has been created, then the Client can invoke methods upon the EJB as if it was a local object. Clients cannot reference, or invoke, and EJB directly. Any calls made to the EJB must be made through the interfaces that the EJB provides. The interface that is used to create a local reference to the EJB is called the *home interface*. When an EJB is deployed in WebSphere, this home interface is made available to Clients by means of a searchable namespace. This means that a Client can look up an address on the namespace. If there is a home interface at that address, and it is the home interface to the EJB that they were looking for, then the Client can create a local instance of that home interface, and then, from that, a local reference to the EJB can be created.

What code is needed in the Client?

The following code fragment illustrates how to Find and Create a local instance of the Inquiry EJB only. The same will need to be done to Find and Create a local copy of the Publish EJB.

```
private com.ibm.uddi.ejb.Inquiry inquiry = null;
// This private variable, "inquiry" is going to be the local reference to
// the EJB in WebSphere declaring it outside the scope of a method means
// that this same reference can be used throughout the client, without
// having to query the namespace again.
public void homeLookup()
{
// These variables simply determine the address of the JNDI namespace, and
// the address of the home interface within that namespace.

// String naming_factory = "com.ibm.ejs.ns.jndi.CNInitialContextFactory";
//WAS 4.0.2 Naming Factory
String naming_factory = "com.ibm.websphere.naming.WsnInitialContextFactory";
//WAS 5.0 Naming Factory

String namespace_address = "iiop://localhost:2809/";
//The address of the namespace
String home_address = "com/ibm/uddi/ejb/InquiryHome";
//The address of the home interface within the JNDI namespace

java.util.Hashtable environment = new java.util.Hashtable();
environment.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY,
                naming_factory);
environment.put(javax.naming.Context.PROVIDER_URL, namespace_address);

try
{
    javax.naming.InitialContext ic =
        new javax.naming.InitialContext(environment);
    // Create a context using the details above to connect to the namespace

    Object o = ic.lookup(home_address);
    // Do a lookup to see if there is an ejb_home at the address specified
    // above

    // Now create a valid home instance for the EJB type we want to create
    com.ibm.uddi.ejb.InquiryHome home =
        (com.ibm.uddi.ejb.InquiryHome)(javax.rmi.PortableRemoteObject.narrow(o,
        com.ibm.uddi.ejb.InquiryHome.class));

    inquiry = home.create();
    // Now create a local reference of the EJB, by using the home.create()
    // method. Any business method that is intended for the EJB in Websphere
    // must be invoked against this inquiry object.
}
catch (javax.naming.NamingException ne) {ne.printStackTrace();}
// This is thrown if there was a problem connecting to the namespace, or
// finding the home_address in the namespace
catch (java.rmi.RemoteException re) {re.printStackTrace();}
// This usually indicates some sort of system failure, either WebSphere is
// not running, or there is a communications problem
catch (javax.ejb.CreateException ce) {ce.printStackTrace();}
// This is thrown if the EJB reference cannot be created from the home
// instance.
}
}
```

Writing Client code to use the EJB API

Once the reference to the EJB has been created (the Inquiry Object, in the above code), then the reference can be treated like any other Java Object. This is an example method using the UDDI EJB API - the only important point to remember is that, although the Inquiry Object has been created as a local reference, it is still

referring to a remote EJB Object in a different server, possibly even in a different country. This means that at the very least a `javax.rmi.RemoteException` must be caught on each method call that is made to the EJB.

```
public void findBusiness()
{
    System.out.println("Find Business:");
        NameList names = new NameList();
        names.add(new Name("IBM Corporation"));
    //Create the list of names to find in the UDDI Registry, here just one is
    // used, "IBM Corporation"
        try
        {
            BusinessList list = inquiry.findBusiness(names);
            //This is the call to the inquiry EJB that searches through the
            // UDDI Registry

    // Now display the amount of business found, and for each one, get the
    // BusinessKey, the BusinessName and the amount of Services that Business
    // has
        System.out.println("There are "+list.getBusinessInfos().size()+
            " matching Businesses in this registry");
            for (int i=0;i<list.getBusinessInfos().size();i++)
            {
                BusinessInfo business = list.getBusinessInfos().get(i);
                System.out.println("\nBusinessKey = "+business.getBusinessKey());
                System.out.println("BusinessName = "
                    + business.getNames().get(0).getNameString());
                System.out.println("This Business Has "
                    + business.getServiceInfos().size()+" Services\n");
            }
        }

    // This is a UDDI specific exception, and will be thrown if for example an
    // invalid name was used as the search criteria
    catch (com.ibm.uddi.datatypes.DispositionReportException e) {
        this.handleDispositionReportException(e);}

        catch (java.rmi.RemoteException re) {re.printStackTrace();}
    // This is the RemoteException that is thrown if there has been a system
    // failure or a connection problem.
}
}
```

What new code is needed on the Client?

Just as each EJB has an interface listed on the JNDI namespace, the `javax.transaction.UserTransaction` class also has an interface listed. This means that the same method used to get a local instance of an EJB can be applied to get a local instance of the `UserTransaction` class. Again, this code can be used to find the `UserTransaction` reference on the namespace, in addition to the code required to find the `Inquiry EJB` and the `Publish EJB`, or, alternatively, there is a slightly more elegant method used in the `TransactionEJBClientSample.java`.

```
public void txLookup()
{
    private javax.transaction.UserTransaction tx = null;
    // This is the private variable that will be used to hold the
    // UserTransaction Object declaring it outside the scope of a method means
    // that this same reference can be used throughout the client, without
    // having to query the namespace again.

    // These variables simply determine the address of the JNDI namespace, and
    // the address of the home interface within that namespace.

    // String naming_factory = "com.ibm.ejs.ns.jndi.CNInitialContextFactory";
    //WAS 4.0.2 Naming Factory
```



```

String naming_factory = "com.ibm.websphere.naming.WsnInitialContextFactory";
//WAS 5.0 Naming Factory

String namespace_address = "iiop://localhost:2809/";
//The address of the namespace
String transaction_address = "jta/usertransaction";
//The address of the UserTransaction interface within the JNDI namespace

java.util.Hashtable environment = new java.util.Hashtable();
environment.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY,
    naming_factory);
environment.put(javax.naming.Context.PROVIDER_URL, namespace_address);

try
{
    javax.naming.InitialContext ic =
        new javax.naming.InitialContext(environment);
        // Create a context using the details above to connect to the namespace
    Object remote_object = ic.lookup(transaction_address);
    // Do a lookup to see if there is a UserTransaction Object at the
    // address specified above
    tx = (javax.transaction.UserTransaction)remote_object;
    //Convert the remote object found into a UserTransaction Object,
    // and assign to the private variable
}
catch (javax.naming.NamingException ne) {ne.printStackTrace();}
// This is thrown if there was a problem connecting to the namespace,
// or finding the transaction_address in the namespace
}

```

Writing Client code to use the EJB API with a Client transaction

To perform an Inquiry, a Publish or a Delete upon the IBM WebSphere UDDI Registry with client side transactional support requires very little additional code compared to doing the same operations without client side transactional support. Using the same code that is listed above (in "Writing Client Code to use the EJB API"), this example illustrates how easy client side transactions are to implement.

The additional lines of code needed are in bold type. This code also assumes that there is a variable called *tx* that has been declared at the class scope.

```

public void findBusiness()
{
//Just as there are UDDI and RMI specific exceptions thrown,
// 5 more exceptions need to be caught.
try
{
tx.begin(); //This begins the transaction context
System.out.println("Find Business:");
NameList names = new NameList();
names.add(new Name("IBM Corporation"));
    // Create the list of names to find in the UDDI Registry, here just one is
    // used, "IBM Corporation"
try
{
BusinessList list = inquiry.findBusiness(names);
    // This is the call to the inquiry EJB that searches through the UDDI Registry

    // Now display the amount of business found, and for each one, get the Business
    // Key, the BusinessName and the amount of Services that Business has
System.out.println("There are "
    +list.getBusinessInfos().size()+" matching Businesses in this registry");
    for (int i=0;i<list.getBusinessInfos().size();i++)
    {
BusinessInfo business = list.getBusinessInfos().get(i);

```

```

System.out.println("\nBusinessKey = "+business.getBusinessKey());
System.out.println("BusinessName = "
+business.getNames().get(0).getNameString());
System.out.println("This Business Has "
+business.getServiceInfos().size()+" Services\n");
}
}
// This is a UDDI specific exception, and will be thrown if for example an
// invalid name was used as the search criteria
catch (com.ibm.uddi.datatypes.DispositionReportException e) {
    this.handleDispositionReportException(e);}
catch (java.rmi.RemoteException re) {re.printStackTrace();}
// This is the RemoteException that is thrown if there has been a system
// failure or a connection problem.

tx.commit(); // This ends the transaction context
}
catch (javax.transaction.NotSupportedException nse) {
    nse.printStackTrace();}
catch (javax.transaction.RollbackException rbe) {
    rbe.printStackTrace();}
catch (javax.transaction.SystemException se) {se.printStackTrace();}
catch (javax.transaction.HeuristicMixedException hme) {
    hme.printStackTrace();}
catch (javax.transaction.HeuristicRollbackException hrbe) {
    hrbe.printStackTrace();}
}

```

- The datatypes package
- Methods in the EJB Interface

Datatypes package in the UDDI Registry

Below is a table listing the classes in the com.ibm.uddi.datatypes package, the elements in the UDDI v2.0 XML schema, and the correspondence between the two.

com.ibm.uddi.datatypes Class	Corresponding UDDIv2.0 XML Schema Element	Notes on DatatypeClass
AccessPoint	accessPoint	
Address	address	
	String addressLine	
AdressLineList		Encapsulates a Vector of addressLine Strings
AddressList		Encapsulates a Vector of Address objects
AssertionStatusItem	assertionStatusItem	
AssertionStatusItemList		Encapsulates a Vector of AssertionStatusItem objects
AssertionStatusReport	assertionStatusReport (response message)	
	String authInfo	
AuthToken		Object containing authInfo String and operator name
	String bindingKey	
BindingDetail	bindingDetail (response message)	
BindingTemplate	bindingTemplate	

BindingTemplateList	bindingTemplates	Encapsulates a Vector of BindingTemplate objects
BusinessDetail	businessDetail (response message)	
BusinessDetailExt	businessDetailExt (Response message)	**
BusinessEntity	businessEntity	
BusinessEntityExt	businessEntityExt	**
BusinessEntityExtList		Encapsulates a Vector of BusinessEntityExt objects **
BusinessEntityList		Encapsulates a Vector of BusinessEntity objects
BusinessInfo	businessInfo	
BusinessInfoList	businessInfo	Encapsulates a Vector of businessInfo objects
	String businessKey	
BusinessList	businessList (response message)	
BusinessService	businessService	
BusinessServiceList	businessServices	Encapsulates a Vector of BusinessService objects
CategoryBag	categoryBag	
	String completionStatus	
Contact	contact	
ContactList	contacts	Encapsulates a Vector of Contact objects
Description	description	
DescriptionList		Encapsulates a Vector of Description objects
DiscoveryUrl	discoveryURL	
DiscoveryUrlList	discoveryURLs	Encapsulates a Vector of DiscoveryURL objects
DispositionReport	dispositionReport	
DispositionreportException		Exception thrown by EJB interface functions when an error occurs
Email	email	
EmailList		Encapsulates a Vector of Email objects
EndPoint		Used as baseclass for AccessPoint and HostingRedirector providing mutual exclusivity
ErrInfo	errInfo	
	findQualifier	
FindQualifier	findQualifiers	
	String fromKey	
HostingRedirector	hostingRedirector	
IdentifierBag	identifierbag	

InquiryOptions		Encapsulates a FindQualifiers object and a maxrows field. Used in find_* API calls to specify search options
InstanceDetails	instanceDetails	
	String instanceParms	
	String keyValue	
KeyedReference	keyedReference	
keysOwned	keysOwned	
LanguageString		Abstract class, extended by some of the datatypes, which represents a string that can optionally be tagged with xml:lang.
Name	name	
NameList		Encapsulates a Vector of Name objects
OverviewDoc	overviewDoc	
	String overviewURL	
	String personName	
Phone	phone	
PhoneList		Encapsulates a Vector of Phone objects
PublisherAssertion	publisherAssertion	
PublisherAssertionList		Encapsulates a Vector of Publisher Assertion objects
PublisherAssertions	publisherAssertions (response message)	
RegisteredInfo	registeredInfo (response message)	
	relatedBusinessInfo	not used
	relatedBusinessInfos	not used
RelatesBusinessesList	relatedBusinessesList	
RelatedBusinessInfo	relatedBusinessInfo	
RelatedBusinessInfos	relatedBusinessInfos	
Result	result	
ResultList		Encapsulates a Vector of Result objects
ServiceDetail	serviceDetail (response message)	
ServiceInfo	serviceInfo	
ServiceInfoList	serviceInfos	Encapsulates a Vector of serviceInfo objects
	String serviceKey	
ServiceList	serviceList (response message)	
	sharedRelationships	not used
SharedRelationships	sharedRelationships	
Tmodel	tModel	
TModelBag	tModelBag	
TModelDetail	tModelDetail (response message)	
TModelInfo	tModelInfo	

TModelInfoList	tModelInfos	Encapsulates a Vector of TModelInfo objects
TModelInstanceInfo	tModelInstanceInfo	
TModelInstanceInfoList	tModelInstanceDetails	Encapsulates a Vector of TModelInstanceInfo objects
	String tModelKey	
TModelList	tModelList (response message)	
TModels		Encapsulates a Vector of TModel objects
	String toKey	
	String uploadRegister	
UploadRegisterList		Encapsulates a Vector of uploadRegister strings

** Used in UDDI API functions relating to BusinessDetailExtension. These UDDI API functions are not implemented in Version 1 of the IBM WebSphere UDDI Registry.

In general, a datatype called *DatatypeList* contains a vector of *Datatype* objects. Often these correspond to XML schema elements with plural names. (For example the datatype *Contact* corresponds to XML element *contact*, and *ContactList* corresponds to *contacts*.) Where there is no "plural" XML schema element for a particular *Datatype*, often there is still a *DatatypeList* where it is useful to have one, e.g. *AddressList*.

The exceptions to this naming convention occur when there is an existing XML schema element ending in "List". The exceptions are: *TModelList*, *ServiceList*, *BusinessList*. In these cases, the corresponding datatypes are given the same names as the XML schema elements, and the datatypes that would have had these names are called: *TModels*, *BusinessServiceList*, *BusinessEntityList*.

EJB interface methods in the UDDI Registry

Inquiry

```
findBinding
findBusiness
findRelatedBusinesses
findService
findTModel
getBindingDetail
getBusinessDetail
getServiceDetail
getTModelDetail
```

Publish

```
addPublisherAssertions
deleteBinding
deleteBusiness
deletePublisherAssertions
deleteService
deleteTModel
getAssertionStatusReport
getRegisteredInfo
getPublisherASsertions
saveBinding
```

```
saveBusiness
saveService
saveTModel
setPublisherAssertions
```

Each method is overloaded and can take various combinations of arguments. The Javadoc contains detailed information about each method.

Note that `get_authToken` and `discard_authToken` are not implemented, as WebSphere security is used instead.

UDDI troubleshooting tips

When the IBM WebSphere UDDI Registry is running, it might issue messages to report events or errors. You can use these messages, described in Messages as your first aid to problem determination. If you need more details about the causes of a problem, you can turn on tracing for UDDI, as described in Turning on UDDI trace.

- Turning on UDDI trace

Below are a few of the common causes of errors that might be found and their suggested solutions.

- If you set up the UDDI Registry application with a JDBC driver and datasource that reference Cloudscape, but set the `persist` property in `uddi.properties` to specify DB2, **or vice versa**, then some unexpected behavior will result, such as a fatal error on deleting an entity. If this happens, you should check that the above details are not in conflict. This only applies to a UDDI Registry installation on a single appserver.
- If you get a message "The application failed to initialize" when trying to access the UDDI User console and you are using DB2 as the persistence store for the UDDI Registry, a likely cause of the problem is that you specified the wrong userid and/or password when you ran the script to install the UDDI Registry application. If this occurs rerun the script ensuring you use the correct userid and password.
- You might find that, after uninstalling and reinstalling the UDDI Registry, you get errors from the UDDI User Console of the form:
"Error 500: JSPG0059E: Unable to compile class for JSP".
If this occurs, then you should clear out the temp directory of the WebSphere AppServer.
- When running one of the UDDI setup scripts `setupuddi.jacl` or `removeuddi.jacl`, if you get an error such as:
WASX7017E: Exception received while running file "setupuddi.jacl"; exception information: com.ibm.bsf.BSFException: error while eval'ing Jacl expression: java.util.MissingResourceException: Can't find resource for bundle java.util.PropertyResourceBundle, key ErrMsgIncorrectNumArgs
then please ensure that the file `setupuddimessages.jafr` is located in the *lib* subdirectory of the WebSphere deployment manager or application server under which you are running the script.
- When running the DB2 Setup Wizard, if you get an error stating "Invalid userid and password", then it could be caused by any of the following situations:
 - You have supplied an invalid userid or password - re-enter with a valid userid and password.
 - The supplied userid does not have the necessary privileges - retry with a userid that has appropriate privileges.
 - DB2 is stopped when you run the Wizard - start DB2 and retry the Wizard.

- The UDDI20 database already exists and has been removed previously and, as such, is not catalogued. The DB2 Wizard does not recognize this situation and gives the error. You now have two options.
 1. If you wish to use the existing database then you will need to catalogue it and there is no need to rerun the Wizard.
 2. If you wish to create a new database you will need to recatalog the database and re-run the DB2 Wizard and choose the option to overwrite the database. (Any existing data **WILL** be lost.)

5.0.1 +

- Catalog the database by:
 - **Windows:**

```
>db2cmd
>db2cat -d uddi20
```
 - **Unix platforms**

```
>su db2inst1 (or name of your db2 instance)
>db2 CATALOG DATABASE UDDI20
```
- **5.0.1 +** ensure that, if you are using a non-English installation of DB2, you have applied PTF1.

Note: **5.0.1 +** If you have a copy of the file SetupDB2UDDI.jar in your appserver directory, then the application of the base and Network Deployment PTFs will not update SetupDB2UDDI.jar in your appserver directory. You must apply the PTF for Network Deployment to your DeploymentManager file structure to update the SetupDB2UDDI.jar located there (in the /IDDIReg/scripts subdirectory), and then manually copy this jar to any appserver you may wish to run it on.

- There is a limitation concerning URL rewriting causing JavaScript syntax errors on several Web pages in the UDDI User Console. Because of this, cookies must be enabled in client browsers, the application server must have cookies enabled as the session tracking mechanism, and URL rewriting must be disabled.
- If you have an existing DB2 version of the UDDI Registry database, and you use the UDDI DB2 setup wizard to replace this database with a new one, and if the database is in use at the time that you run the UDDI DB2 setup wizard, then the existing database is not overwritten.
- When running the UDDI DB2 setup wizard, as part of the installation step "Setting up the UDDI Registry to use DB2 within a deployment manager cell" or "Setting up the UDDI Registry to use DB2 in a single appserver", in addition to running the *was_install\bin\setupcmdline.bat* directory, you should also type either *set PATH=%WAS_PATH%* (for Windows platforms) or *export PATH=/opt/WebSphere/AppServer/java/bin:\$PATH* (for Unix platforms) to ensure that you have access to Java.
- UDDI user console "Page cannot be displayed" errors with Internet Explorer. If you use Internet Explorer with the option "Show friendly HTTP error messages" enabled and you have WebSphere Application Server security enabled (user ID and password authentication enabled), you might experience intermittent errors on the browser, such as "Page cannot be displayed", when navigating the UDDI user console. This might be particularly noticeable when accessing the publish actions. To avoid such errors, disable the "Show friendly HTTP error messages" option on Internet Explorer. This option is found under **Tools > Internet options > Advanced Tab > Browsing Section**
- When using SOAP or UDDI4J, it is sometimes necessary to call `setServiceKey("")` before saving your changes, except with the EJB interface where this might result in an error.

- There are known problems with inquiries issued against the UDDI Registry if IBM Cloudscape is used as the persistence store for the registry data. Certain complex inquiries might produce unexpected results, or could fail. If your application needs to make inquiries of this nature, consider using DB2 as the persistence store. Note: DB2 must be used for production purposes. The IBM Cloudscape support is only provided for development and test use.
- You might see errors if you specify requests which specify more than 5 category values, more than 5 identifier values, or more than 5 technical model (tModel) values.
- If you stop and restart the UDDI Registry application from the administrative console, and then try to access the Registry through the user console, an "Error 500 - object is not an instance of a declaring class" displays on the user console, and the error message "SRVE0026E" displays in the system log. You cannot access the UDDI Registry until you restart the WebSphere Application Server. To avoid this problem when restarting the UDDI Registry, you should set *Prefer WEB-INF classes* on the panel navigated to by the following steps: **Applications > Manage Applications > UDDIRegistry > Web Modules defined for this Application > gui.war.**
- It is possible that a scripting error displays when you are running the **wsadmin appserversetupuddi.jacl** command.

During installation, if you see the following error at the end of running the **appserversetupuddi.jacl** command, you can safely ignore the error. It is recommended that you start (or stop and restart) the application server and then continue.

Here is an example of the error:

```
UDIN2041I: Starting UDDI application. UDIN8019E: startApplication command for
  appname caught exception Exc. Values are: appname=UDDIRegistry,
  Exc=com.ibm.ws.scripting.Scripting
  Exception: com.ibm.websphere.management.exception.Connector
  Exception: ADMC0009E: Failed to make the SOAP RPC call: invoke
```

- If you find that the UDDI User Console is giving unexpected results, in particular if you have installed a FixPack but are not seeing all of the fixes, then you should clear out the compiled UDDI JSP files from the temp directory of the WebSphere Application Server. These files will be in a directory identified by the name of the UDDI Registry located beneath the temp directory.
- If attempting to use a remote DB2 database and you are experiencing problems attaching to the remote system, one of the possible causes might be IP addressing. You should not have this problem if the remote system is using a static IP address. If, however, the remote system is using DHCP, the two systems must be aware of each others subnet mask.

For Windows, the subnet mask can be found by starting a Command Prompt and typing "ipconfig" on the remote system. On the host system, the WINS might need to be edited to add the remote subnet mask. To do this on Windows go to the following commands:

1. START => Network and Dial-up Connections => Local Area Network Connection 2 => Internet Protocol (TCP/IP) and click on Properties
2. Click on "Advanced" on the next screen
3. Click on the WINS tab on the next screen and add the new subnet mask
4. Move the new subnet mask to the top of the list by highlighting it and pressing the "Up" arrow until it is the top of the list of WINS addresses

On Unix platforms, you can use *ifconfig* to determine the subnet mask.

Turning on UDDI trace

You can enable UDDI-specific trace in the same way as you enable other tracing in the WebSphere Application Server.

The following is a list of trace strings that may be used:

- com.ibm.uddi.api
- com.ibm.uddi.config
- com.ibm.uddi.datatypes
- com.ibm.uddi.dom
- com.ibm.uddi.ejb
- com.ibm.uddi.exception
- com.ibm.uddi.exceptions
- com.ibm.uddi.gui
- com.ibm.uddi.gui.inquire
- com.ibm.uddi.gui.publish
- com.ibm.uddi.persistence
- com.ibm.uddi.persistence.jdbc
- com.ibm.uddi.persistence.jdbc.cloudscape
- com.ibm.uddi.persistence.jdbc.db2
- com.ibm.uddi.ras
- com.ibm.uddi.security
- com.ibm.uddi.soap
- com.ibm.uddi.uuid
- com.ibm.uddi.validation
- com.ibm.uddi.xml

For example, to trace the UDDI User Console you would specify:

```
'com.ibm.uddi.gui=all=enabled'
```

This would enable all types of trace for the gui. Please refer to "Enabling trace" elsewhere in the WebSphere Application Server V5.0 InfoCenter for more information about using the administrator console to enable/disable trace.

Messages

When the IBM WebSphere UDDI Registry is running, it might issue messages to report events or errors. The messages are in the form `UDxxnnnns` where:

xx is a two character descriptor identifying which component is involved
nnnn give the error code being issued
s is either I (Information) or E (Error)

The prefix `UDxxnnnns`: is followed by text which describes the event or error. For some messages, the first word of the text is one of the form (MSN=SSSS). The SSSS provides a message sequence number (or MSN), which identifies the unique circumstance in which the message was issued, and is of use where the same message can be issued in more than one circumstance.

To help you diagnose problems and minimize the need to enable trace in any of the above components, view the messages table. You can view the messages by prefix or component, whichever is easiest for you to find in the table. All messages are documented with user/system action and explanation.

The text for the UDDI messages is contained in a file `uddiresourcebundles.jar` which is placed, by the installation process, into the `\lib` subdirectory (Windows) of the

WebSphere application server into which the UDDI Registry was installed. If you will be running a console or log analyzer from another process; for example, to analyze the activity log, then you must place a copy of *uddiresourcebundles.jar* into a directory which is within the classpath of that process. Otherwise, the message lookup for the UDDI messages will fail.

UDDI Components Message Prefix Table	
click on individual links for message documentation for the component	
UDAI	API
UDCF	Configuration
UDDA	Datatypes
UDDM	DOM
UDEJ	EJB Interface
UDEX	Exceptions
UDIN	Installation
UDLC	Local API
UDPR	Persistence
UDRS	Logging
UDSC	Security
UDSP	SOAP Interface
UDUC	User Console
UDUT	UDDI Utility Tool
UDUU	UUID

UDAI (Web Services UDDI) messages

There are no messages issued by this component.

UDCF (Web Services UDDI) messages

UDCF0001E: Exception occurred while getting int value of configuration property "<property>", exception: "<exception>"

Explanation: This message is issued when an attempt to read the value of a configuration property from the `uddi.properties` file and convert it to integer has failed with the indicated exception.

User Response: Check that the `uddi.properties` file contains a value for the indicated configuration property, and that the value is valid. Check also that the indicated configuration property is a legal property. Refer to the InfoCenter for further information about global configuration properties and the `uddi.properties` file.

UDCF0002E: Exception occurred while getting long value of configuration property "<property>", exception: "<exception>"

Explanation: This message is issued when an attempt to read the value of a configuration property from the `uddi.properties` file and convert it to long has failed with the indicated exception.

User Response: Check that the `uddi.properties` file contains a value for the indicated configuration property, and that the value is valid. Check also that the indicated configuration property is a legal property. Refer to the InfoCenter for further information about global configuration properties and the `uddi.properties` file.

UDCF0003E: Exception occurred while getting boolean value of configuration property "<property>", exception: "<exception>"

Explanation: This message is issued when an attempt to read the value of a configuration property from the uddi.properties file and convert it to boolean has failed with the indicated exception

User Response: Check that the uddi.properties file contains a value for the indicated configuration property, and that the value is valid. Check also that the indicated configuration property is a legal property. Refer to the InfoCenter for further information about global configuration properties and the uddi.properties file.

UDCF0004E: Failed to load UDDI global properties file.

Explanation: This message is issued when the UDDI global configuration properties file, uddi.properties, cannot be loaded. Default values for the global configuration properties will be set, but these defaults may not be adequate for many of the properties, so you should investigate and resolve this problem.

User Response: Check that the uddi.properties file exists and is in the correct directory. Refer to the InfoCenter for further information about global configuration properties and the uddi.properties file.

UDCF0005E: Exception occurred while loading UDDI global configuration properties, exception: "<exception>"

Explanation: This message is issued when an attempt to load the UDDI global configuration properties from the uddi.properties has failed with the indicated exception. Default values for the global configuration properties will be set, but these defaults may not be adequate for many of the properties, so you should investigate and resolve this problem.

User Response: Check that the uddi.properties file exists and contains valid values for each of the configuration properties. Refer to the InfoCenter for further information about global configuration properties and the uddi.properties file.

UDDA (Web Services UDDI) messages

There are no messages issued by this component.

UDDM (Web Services UDDI) messages

There are no messages issued by this component.

UDEJ (Web Services UDDI) messages

There are no messages issued by this component.

UDEX (Web Services UDDI) messages

There are no messages issued by this component.

UDIN (Web Services UDDI) messages

UDIN0001I: Assuming hard coded defaults.

Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None..

UDIN0002I: Cloudscape classpath is clpath. Value is:

Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0003I: Looking for childtype childname under parenttype parentname. Values are:
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0004I: Looking for childtype childname under parenttype parentname and parenttype2 parentname2. Values are:
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0005I: Conflict found with existing childtype childname. Values are:
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0006I: Not creating requested childtype. Value is:
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0007I: Seeking parenttype with requested id of parentname. Values are:
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0008I: Seeking parenttype with requested id of parentname under parenttype2 parentname2. Values are:
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0009I: Attempting to create childtype under parenttype of parentID. Values are:
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0010I: Create command that will be issued is:
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0011I: childtype childId was successfully created. Values are:
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0012I: Looking for builtin_rra.
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0013I: List for J2CResourceAdapter returned N members. Value is:
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0014I: Hunting J2CResourceAdapter associated with Node nodename.

Value is:

Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0015I: Using rraID as builtin_rra. Value is:

Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0016I: Using provider class of implclass with a classpath of clpath. Values are:

Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0017I: Installing to server servername, node nodename using database type of dbtype. Values are:

Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0018I: Attempting to create UDDI JDBCProvider.

Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0019I: Attempting to create UDDI Datasource.

Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0020I: Application Manager appmgr found. Value is:

Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0021I: Attempting to install UDDI Registry application.

Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0022I: Checking for installed UDDI Registry application of name appname. Value is:

Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0023W: Application of name appname is not present. Value is:

Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0024I: ApplicationManager not running, so application will not need to be stopped.

Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

- UDIN0025I: Stopping application of name appname. Value is:**
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.
User Response: None.
- UDIN0026W: stopApplication command for application appname caught exception Exc. Application might not have been running on this server. Values are:**
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.
User Response: None.
- UDIN0027I: Application appname stopped successfully. Value is:**
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.
User Response: None.
- UDIN0028I: Removing application appname. Value is:**
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.
User Response: None.
- UDIN0029I: Application appname removed successfully. Value is:**
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.
User Response: None.
- UDIN0030I: Adding resource bundles to repository.**
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.
User Response: None.
- UDIN0031I: Adding Cloudscape user functions to repository.**
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.
User Response: None.
- UDIN0032I: UDDI configuration properties file already exists. Only the persister and getServletURLPrefix properties will be overwritten.**
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.
User Response: None.
- UDIN0033I: Editing UDDI configuration properties file propsfile. Value is:**
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.
User Response: None.
- UDIN0034I: Url prefix found. Updating it to discoveryURL. Value is:**
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.
User Response: None.
- UDIN0035I: Persister property found. Updating it to dbtype. Value is:**
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.
User Response: None.
- UDIN0036I: Adding UDDI configuration properties file to repository for cell cellname under target node and server. Value is:**
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0037I: ws.ext.dir processing starting.
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0038I: serverID is:
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0039I: JVM is:
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0040I: Out of N properties we located M matches at positions poslist.
Values are:
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0041I: Building new ws.ext.dirs properties.
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0042I: SYSPROP is:
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0043I: ws.ext.dir has been set with new sysprop. Value is:
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0044I: ws.ext.dir update skipped, required changes already present.
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0045I: ws.ext.dir processing step complete.
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0046I: Cleaning up temporary version of properties file temppropsfile.
Value is:
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0047I: Issuing nodeSync.
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

- UDIN0048I: UDDI Registry successfully installed. Please restart server servename to activate configuration changes. Value is:**
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.
User Response: None.
- UDIN0049I: Application Manager appmgr found. Value is:**
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.
User Response: None.
- UDIN0050I: Server is not running, so will not need to be stopped.**
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.
User Response: None.
- UDIN0051I: Stopping server servename under node nodename. Values are:**
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.
User Response: None.
- UDIN0052I: Server servename stopped successfully. Value is:**
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.
User Response: None.
- UDIN0053I: Restarting application server**
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.
User Response: None.
- UDIN0054I: Application server servename restarted successfully. Value is:**
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.
User Response: None.
- UDIN0055I: Please ignore any errors concerning the serverStartupSyncEnabled attribute.**
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.
User Response: None.
- UDIN0101I: Attempting to save new configuration.**
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.
User Response: None.
- UDIN0102I: Changes saved successfully.**
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.
User Response: None.
- UDIN0103I: Changes were not saved on this call.**
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.
User Response: None.
- UDIN0104I: Attempting to save new configuration.**
Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.
User Response: None.

UDIN0105I: Changes saved successfully.

Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0106I: Attempting to save ws.ext.dir changes.

Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0107I: Changes saved successfully.

Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0108I: Attempting final save of new configuration.

Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN0109I: Changes saved successfully.

Explanation: This is an informational message issued by the UDDI setup script setupuddi.jacl.

User Response: None.

UDIN1001I: Application Manager appmgr found. Value is:

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1002I: Server is not running, so will not need to be stopped.

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1003I: Stopping server servername under node nodename. Values are:

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1004I: Server servername stopped successfully. Value is:

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1005I: Resource bundles file will be removed from repository if present.

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1006I: Removing resource bundles from repository.

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1007I: Resource bundles successfully removed from repository.

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1008I: Cloudscape user functions file will be removed from repository if present.

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1009I: Removing Cloudscape user functions from repository.

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1010I: Cloudscape user functions successfully removed from repository.

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1011I: Application Manager appmgr found. Value is:

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1012I: Checking for installed UDDI Registry application of name appname. Value is:

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1013W: Application of name appname is not present. Value is:

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1014I: ApplicationManager not running, so application will not need to be stopped.

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1015I: Stopping application of name appname. Value is:

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1016W: stopApplication command for application appname caught exception Exc. Application might not have been running on this server. Values are:

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1017I: Application appname stopped successfully. Value is:

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1018I: Removing application appname. Value is:

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1019I: Application appname removed successfully. Value is:

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1020I: UDDI datasource will be removed from server servername in node nodename if present. Values are:

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1021I: Removing UDDI datasource.

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1022I: UDDI datasource successfully removed.

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1023I: UDDI JDBC driver will be removed from server servername in node nodename if present. Values are:

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1024I: Removing UDDI JDBC driver.

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1025I: UDDI JDBC driver successfully removed.

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1026I: UDDI configuration properties file will be removed from repository if present.

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1027I: Removing configuration properties file from cell cellname, node nodename and server servername. Values are:

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1028I: Configuration properties file successfully removed from repository.

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1029I: Issuing nodeSync.

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1030I: UDDI Registry application, JDBC driver and datasource removed successfully.

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1031I: Restarting application server.

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1032I: Application server servername restarted successfully. Value is:

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1033I: Please ignore any errors concerning the serverStartupSyncEnabled attribute.

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1034I: ws.ext.dir processing starting.

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1035I: serverID is:

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1036I: JVM is:

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1037I: Out of N properties we located M matches at positions poslist.

Values are:

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1038I: Removing UDDI values from ws.ext.dirs properties.

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1039I: ws.ext.dir has been set with new sysprop. Value is:

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1040I: ws.ext.dir update skipped, required changes already present.

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.

UDIN1041I: ws.ext.dir processing step complete.

Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.
UDIN1101I: Attempting to save new configuration.
Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.
UDIN1102I: Changes saved successfully.
Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.
UDIN1103I: Attempting to save new configuration.
Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.
UDIN1104I: Changes saved successfully.
Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.
UDIN1105I: Attempting to save new configuration.
Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.
UDIN1106I: Changes saved successfully.
Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.
UDIN1107I: Attempting final save of new configuration.
Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.
UDIN1108I: Changes saved successfully.
Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.
UDIN1109I: Attempting to save ws.ext.dir changes.
Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.
UDIN1110I: Changes saved successfully.
Explanation: This is an informational message issued by the UDDI setup script removeuddi.jacl.

User Response: None.
UDIN2001I: Assuming hard coded defaults.
Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.
UDIN2002I: Listing members of type parenttype. Value is:
Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

- UDIN2003I: List for type parenttype returned N members. Values are:**
Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.
User Response: None.
- UDIN2004I: Seeking parenttype with requested id of parentname. Values are:**
Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.
User Response: None.
- UDIN2005I: Checking parentID with parentname. Values are:**
Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.
User Response: None.
- UDIN2006I: Using this as parenttype of parentname. Values are:**
Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.
User Response: None.
- UDIN2007I: Checking for existing childtype under parentname. Values are:**
Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.
User Response: None.
- UDIN2008I: List for childtype returned N members. Values are:**
Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.
User Response: None.
- UDIN2009I: No existing childtype present. Value is:**
Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.
User Response: None.
- UDIN2010I: N existing objects of type childtype found, examining for conflict with childname. Values are:**
Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.
User Response: None.
- UDIN2011I: Checking childID with name childname. Values are:**
Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.
User Response: None.
- UDIN2012I: Conflict found with existing childtype of id childID. Values are:**
Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.
User Response: None.
- UDIN2013I: Not creating requested object of type childtype. Value is:**
Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.
User Response: None.
- UDIN2014I: Conflict found with existing childtype, removing existing childtype. Value is:**
Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.
User Response: None.

UDIN2015I: Removal of childtype was successful. Value is:

Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2016I: Not in conflict.

Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2017I: Attempting to create childtype under parentname of parentID.

Values are:

Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2018I: Create command that will be issued is:

Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2019I: childtype childID was successfully created. Values are:

Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2020I: No matches found.

Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2021I: Looking for builtin_rra.

Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2022I: List for J2CResourceAdapter returned N members. Value is:

Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2023I: Hunting J2CResourceAdapter associated with Node nodename.

Value is:

Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2024I: Using rraID as builtin_rra. Value is:

Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2025I: Using provider class of implclass with a classpath of clpath. Values

are: **Explanation:** This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2026I: Installing to node nodename using database type of dbtype. Values

are: **Explanation:** This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2027I: Attempting to create UDDI JDBCProvider.

Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2028I: Attempting to create UDDI Datasource.

Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2029I: Application Manager appmgr found. Value is:

Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2030I: Attempting to install UDDI Registry application.

Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2031I: Checking for installed UDDI Registry application of name appname. Value is:

Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2032I: List for Applications returned N members. Value is:

Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2033W: Application of name appname is not present. Value is:

Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2034I: ApplicationManager not running, so application will not need to be stopped.

Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2035I: Stopping application of name appname. Value is:

Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2036W: stopApplication command for application appname caught exception Exc. Application might not have been running on this server. Values are:

Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2037I: Application appname stopped successfully. Value is:

Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2038I: Removing application appname. Value is:

Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2039I: Application appname removed successfully. Value is:
Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2040I: Attempting to install application appname. Value is:
Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2041I: Starting UDDI application.
Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2042I: Application appname started successfully. Value is:
Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2101I: Attempting to save new configuration.
Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2102I: Changes saved successfully.
Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2103I: Changes were not saved on this call.
Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2104I: Attempting to save post installation configuration.
Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2105I: Changes saved successfully for UDDI Registry.
Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2106I: Attempting to save new configuration.
Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN2107I: Changes saved successfully for UDDI Registry.
Explanation: This is an informational message issued by the UDDI setup script appserversetupuddi.jacl.

User Response: None.

UDIN3001I: Application Manager appmgr found. Value is:
Explanation: This is an informational message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response: None.

UDIN3002I: Checking for installed UDDI Registry application.

Explanation: This is an informational message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response: None.

UDIN3003I: List for Applications returned N members. Value is:

Explanation: This is an informational message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response: None.

UDIN3004W: Application of name appname is not present. Value is:

Explanation: This is an informational message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response: None.

UDIN3005I: ApplicationManager not running, so application will not need to be stopped.

Explanation: This is an informational message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response: None.

UDIN3006I: Stopping application of name appname. Value is:

Explanation: This is an informational message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response: None.

UDIN3007W: stopApplication command for application appname caught exception Exc. Application might not have been running on this server. Values are:

Explanation: This is an informational message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response: None.

UDIN3008I: Application appname stopped successfully. Value is:

Explanation: This is an informational message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response: None.

UDIN3009I: Removing application appname. Value is:

Explanation: This is an informational message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response: None.

UDIN3010I: Application appname removed successfully. Value is:

Explanation: This is an informational message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response: None.

UDIN3011I: UDDI datasource will be removed from server servername in node nodename if present. Values are:

Explanation: This is an informational message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response: None.

UDIN3012I: Removing UDDI datasource.

Explanation: This is an informational message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response: None.

UDIN3013I: UDDI datasource successfully removed.

Explanation: This is an informational message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response: None.

UDIN3014I: UDDI JDBC driver will be removed from server servername in node nodename if present. Values are:
Explanation: This is an informational message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response: None.

UDIN3015I: Removing UDDI JDBC driver from node nodename. Value is:
Explanation: This is an informational message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response: None.

UDIN3016I: UDDI JDBC driver successfully removed.
Explanation: This is an informational message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response: None.

UDIN3017I: UDDI Registry application, JDBC driver and datasource removed successfully.
Explanation: This is an informational message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response: None.

UDIN3101I: Attempting to save new configuration.
Explanation: This is an informational message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response: None.

UDIN3102I: Changes to remove UDDI Registry have been saved successfully.
Explanation: This is an informational message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response: None.

UDIN3103I: Attempting to save new configuration.
Explanation: This is an informational message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response: None.

UDIN3104I: Changes saved successfully.
Explanation: This is an informational message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response: None.

UDIN3105I: Attempting final save of new configuration.
Explanation: This is an informational message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response: None.

UDIN3106I: Changes saved successfully.
Explanation: This is an informational message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response: None.

UDIN6001E: This script must be run in a Deployment Manager environment.
Explanation: This is an error message issued by the UDDI setup script setupuddi.jacl.

User Response:This message is self-explanatory.

UDIN6002E: To install in a standalone application server, use appserversetupuddi.jacl.

Explanation: This is an error message issued by the UDDI setup script setupuddi.jacl.

User Response:This message is self-explanatory.

UDIN6003E: Incorrect number of arguments passed to script.

Explanation: This is an error message issued by the UDDI setup script setupuddi.jacl.

User Response:This message is self-explanatory.

UDIN6004E: Usage is:

Explanation: This is an error message issued by the UDDI setup script setupuddi.jacl. The text following 'Usage is:' gives the syntax for calling setupuddi.jacl.

User Response:This message is self-explanatory.

UDIN6005E: (<db2userid> <db2password> <db2ziplocation> are only required if setting up to use DB2).

Explanation: This is an error message issued by the UDDI setup script setupuddi.jacl.

User Response:This message is self-explanatory.

UDIN6006E: Use all forward (/) slashes to avoid problems with escaping back (\\) slashes.

Explanation: This is an error message issued by the UDDI setup script setupuddi.jacl.

User Response:This message is self-explanatory.

UDIN6007E: Removal of childtype childname caught exception Exc. Values are:

Explanation: This is an error message issued by the UDDI setup script setupuddi.jacl.

User Response:This message is self-explanatory.

UDIN6008E: An exception Exc occurred while creating childtype. Values are:

Explanation: This is an error message issued by the UDDI setup script setupuddi.jacl.

User Response:This message is self-explanatory.

UDIN6009E: Unable to find requested parenttype of parentname. Values are:

Explanation: This is an error message issued by the UDDI setup script setupuddi.jacl.

User Response:This message is self-explanatory.

UDIN6010E: List command for J2CResourceAdapter caught exception Exc. Value is:

Explanation: This is an error message issued by the UDDI setup script setupuddi.jacl.

User Response:This message is self-explanatory.

UDIN6011E: No J2CResourceAdapter objects available.

Explanation: This is an error message issued by the UDDI setup script setupuddi.jacl.

User Response:This message is self-explanatory.

UDIN6012E: An error occurred during execution of setupuddi.jacl. Please check the parameters and try again.

Explanation: This is an error message issued by the UDDI setup script setupuddi.jacl.

User Response:This message is self-explanatory.

UDIN6013E: Uninstall of application appname caught exception Exc. Values are:
Explanation: This is an error message issued by the UDDI setup script setupuddi.jacl.
User Response:This message is self-explanatory.

UDIN6014E: Install of UDDI application caught exception Exc. Value is:
Explanation: This is an error message issued by the UDDI setup script setupuddi.jacl.
User Response:This message is self-explanatory.

UDIN6015E: Could not get JVM.
Explanation: This is an error message issued by the UDDI setup script setupuddi.jacl.
User Response:This message is self-explanatory.

UDIN6016E: Cannot find nodeSync MBean.
Explanation: This is an error message issued by the UDDI setup script setupuddi.jacl.
User Response:This message is self-explanatory.

UDIN6017E: nodeSync failed. UDDI Application may not be fully installed.
Explanation: This is an error message issued by the UDDI setup script setupuddi.jacl.
User Response:This message is self-explanatory.

UDIN6018E: stopServer command for server servername caught exception Exc. Values are:
Explanation: This is an error message issued by the UDDI setup script setupuddi.jacl.
User Response:This message is self-explanatory.

UDIN6019E: startServer command for server servername caught exception Exc. Values are:
Explanation: This is an error message issued by the UDDI setup script setupuddi.jacl.
User Response:This message is self-explanatory.

UDIN6101E: Error saving configuration, changes not saved due to exception Exc. Value is:
Explanation: This is an error message issued by the UDDI setup script setupuddi.jacl.
User Response:This message is self-explanatory.

UDIN6102E: Error saving configuration, changes not saved due to exception Exc. Value is:
Explanation: This is an error message issued by the UDDI setup script setupuddi.jacl.
User Response:This message is self-explanatory.

UDIN6103E: Error saving configuration, changes not saved due to exception Exc. Value is:
Explanation: This is an error message issued by the UDDI setup script setupuddi.jacl.
User Response:This message is self-explanatory.

UDIN6104E: Error saving configuration, changes not saved due to exception Exc. Value is:
Explanation: This is an error message issued by the UDDI setup script setupuddi.jacl.
User Response:This message is self-explanatory.

UDIN7001E: This script must be run in a Deployment Manager environment.

Explanation: This is an error message issued by the UDDI setup script removeuddi.jacl.

User Response:This message is self-explanatory.

UDIN7002E: To remove from a standalone application server, use appserverremoveuddi.jacl.

Explanation: This is an error message issued by the UDDI setup script removeuddi.jacl.

User Response:This message is self-explanatory.

UDIN7003E: Incorrect number of arguments passed to script.

Explanation: This is an error message issued by the UDDI setup script removeuddi.jacl.

User Response:This message is self-explanatory.

UDIN7004E: Usage is:

Explanation: This is an error message issued by the UDDI setup script removeuddi.jacl. The text following 'Usage is:' gives the syntax for calling removeuddi.jacl.

User Response:This message is self-explanatory.

UDIN7005E: stopServer command for server servername caught exception Exc. Values are:

Explanation: This is an error message issued by the UDDI setup script removeuddi.jacl.

User Response:This message is self-explanatory.

UDIN7006E: Removal of resource bundles caught exception Exc. Value is:

Explanation: This is an error message issued by the UDDI setup script removeuddi.jacl.

User Response:This message is self-explanatory.

UDIN7007E: Removal of Cloudscape user functions caught exception Exc. Value is:

Explanation: This is an error message issued by the UDDI setup script removeuddi.jacl.

User Response:This message is self-explanatory.

UDIN7008E: Uninstall of application appname caught exception Exc. Values are:

Explanation: This is an error message issued by the UDDI setup script removeuddi.jacl.

User Response:This message is self-explanatory.

UDIN7009E: Removal of UDDI datasource caught exception Exc. Value is:

Explanation: This is an error message issued by the UDDI setup script removeuddi.jacl.

User Response:This message is self-explanatory.

UDIN7010E: Removal of UDDI JDBC driver caught exception Exc. Value is:

Explanation: This is an error message issued by the UDDI setup script removeuddi.jacl.

User Response:This message is self-explanatory.

UDIN7011E: Removal of configuration properties file caught exception Exc. Value is:

Explanation: This is an error message issued by the UDDI setup script removeuddi.jacl.

User Response:This message is self-explanatory.

UDIN7012E: Cannot find nodeSync MBean.

Explanation: This is an error message issued by the UDDI setup script removeuddi.jacl.

User Response:This message is self-explanatory.

UDIN7013E: nodeSync failed. UDDI Application may not be fully uninstalled.

Explanation: This is an error message issued by the UDDI setup script removeuddi.jacl.

User Response:This message is self-explanatory.

UDIN7014E: startServer command for server servername caught exception Exc.

Values are:

Explanation: This is an error message issued by the UDDI setup script removeuddi.jacl.

User Response:This message is self-explanatory.

UDIN7015E: Could not get JVM.

Explanation: This is an error message issued by the UDDI setup script removeuddi.jacl.

User Response:This message is self-explanatory.

UDIN7101E: Error saving configuration, changes not saved due to exception Exc.

Value is:

Explanation: This is an error message issued by the UDDI setup script removeuddi.jacl.

User Response:This message is self-explanatory.

UDIN7102E: Error saving configuration, changes not saved due to exception Exc.

Value is:

Explanation: This is an error message issued by the UDDI setup script removeuddi.jacl.

User Response:This message is self-explanatory.

UDIN7103E: Error saving configuration, changes not saved due to exception Exc.

Value is:

Explanation: This is an error message issued by the UDDI setup script removeuddi.jacl.

User Response:This message is self-explanatory.

UDIN7104E: Error saving configuration, changes not saved due to exception Exc.

Value is:

Explanation: This is an error message issued by the UDDI setup script removeuddi.jacl.

User Response:This message is self-explanatory.

UDIN7105E: Error saving configuration, changes not saved due to exception Exc.

Value is:

Explanation: This is an error message issued by the UDDI setup script removeuddi.jacl.

User Response:This message is self-explanatory.

UDIN8001E: This script must be run on a standalone application server.

Explanation: This is an error message issued by the UDDI setup script appserversetupuddi.jacl.

User Response:This message is self-explanatory.

UDIN8002E: To install in a Deployment Manager Environment, use setupuddi.jacl.

Explanation: This is an error message issued by the UDDI setup script appserversetupuddi.jacl.

- User Response:**This message is self-explanatory.
- UDIN8003E: Incorrect number of arguments passed to script.**
Explanation: This is an error message issued by the UDDI setup script appserversetupuddi.jacl.
- User Response:**This message is self-explanatory.
- UDIN8004E: Usage is:**
Explanation: This is an error message issued by the UDDI setup script appserversetupuddi.jacl. The text following 'Usage is:' gives the syntax for calling appserversetupuddi.jacl.
- User Response:**This message is self-explanatory.
- UDIN8005E: (<db2userid> <db2password> <db2ziplocation> are only required if setting up to use DB2).**
Explanation: This is an error message issued by the UDDI setup script appserversetupuddi.jacl.
- User Response:**This message is self-explanatory.
- UDIN8006E: Use all forward ("/) slashes to avoid problems with escaping back ("\\) slashes.**
Explanation: This is an error message issued by the UDDI setup script appserversetupuddi.jacl.
- User Response:**This message is self-explanatory.
- UDIN8007E: List command for type parenttype caught exception Exc. Values are:**
Explanation: This is an error message issued by the UDDI setup script appserversetupuddi.jacl.
- User Response:**This message is self-explanatory.
- UDIN8008E: No objects of type parenttype available. Value is:**
Explanation: This is an error message issued by the UDDI setup script appserversetupuddi.jacl.
- User Response:**This message is self-explanatory.
- UDIN8009E: List command for childtype caught exception Exc. Values are:**
Explanation: This is an error message issued by the UDDI setup script appserversetupuddi.jacl.
- User Response:**This message is self-explanatory.
- UDIN8001E: This script must be run on a standalone application server.**
Explanation: This is an error message issued by the UDDI setup script appserversetupuddi.jacl.
- User Response:**This message is self-explanatory.
- UDIN8010E: Error during remove of existing childtype, exception Exc caught. Values are:**
Explanation: This is an error message issued by the UDDI setup script appserversetupuddi.jacl.
- User Response:**This message is self-explanatory.
- UDIN8011E: Create command for childtype caught exception Exc. Values are:**
Explanation: This is an error message issued by the UDDI setup script appserversetupuddi.jacl.
- User Response:**This message is self-explanatory.
- UDIN8012E: Unable to find requested parenttype of parentname. Values are:**
Explanation: This is an error message issued by the UDDI setup script appserversetupuddi.jacl.
- User Response:**This message is self-explanatory.

UDIN8013E: List command for J2CResourceAdapter caught exception Exc. Value is:
Explanation: This is an error message issued by the UDDI setup script appserversetupuddi.jacl.

User Response:This message is self-explanatory.

UDIN8014E: No J2CResourceAdapter objects available.

Explanation: This is an error message issued by the UDDI setup script appserversetupuddi.jacl.

User Response:This message is self-explanatory.

UDIN8015E: An error occurred during execution of appserversetupuddi.jacl. Please check the parameters and try again.

Explanation: This is an error message issued by the UDDI setup script appserversetupuddi.jacl.

User Response:This message is self-explanatory.

UDIN8016E: List command for Applications caught exception Exc. Value is:

Explanation: This is an error message issued by the UDDI setup script appserversetupuddi.jacl.

User Response:This message is self-explanatory.

UDIN8017E: Uninstall of application appname caught exception Exc. Values are:

Explanation: This is an error message issued by the UDDI setup script appserversetupuddi.jacl.

User Response:This message is self-explanatory.

UDIN8018E: Install of UDDI application caught exception Exc. Value is:

Explanation: This is an error message issued by the UDDI setup script appserversetupuddi.jacl.

User Response:This message is self-explanatory.

UDIN8019E: startApplication command for appname caught exception Exc. Values are:

Explanation: This is an error message issued by the UDDI setup script appserversetupuddi.jacl.

User Response:This message is self-explanatory.

UDIN8101E: Error saving configuration, changes not saved due to exception Exc. Value is:

Explanation: This is an error message issued by the UDDI setup script appserversetupuddi.jacl.

User Response:This message is self-explanatory.

UDIN8102E: Error saving configuration, changes not saved due to exception Exc. Value is:

Explanation: This is an error message issued by the UDDI setup script appserversetupuddi.jacl.

User Response:This message is self-explanatory.

UDIN8103E: Error saving configuration, changes not saved due to exception Exc. Value is:

Explanation: This is an error message issued by the UDDI setup script appserversetupuddi.jacl.

User Response:This message is self-explanatory.

UDIN9001E: This script must be run on a standalone application server.

Explanation: This is an error message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response:This message is self-explanatory.

UDIN9002E: To remove from a deployment manager environment, use removeuddi.jacl.

Explanation: This is an error message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response:This message is self-explanatory.

UDIN9003E: Incorrect number of arguments passed to script.

Explanation: This is an error message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response:This message is self-explanatory.

UDIN9004E: Usage is:

Explanation: This is an error message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response:This message is self-explanatory.

UDIN9005E: List command for Applications caught exception Exc. Value is:

Explanation: This is an error message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response:This message is self-explanatory.

UDIN9006E: Uninstall of application appname caught exception Exc. Values are:

Explanation: This is an error message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response:This message is self-explanatory.

UDIN9007E: Removal of UDDI datasource caught exception Exc. Value is:

Explanation: This is an error message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response:This message is self-explanatory.

UDIN9008E: Removal of UDDI JDBC driver caught exception Exc. Value is:

Explanation: This is an error message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response:This message is self-explanatory.

UDIN9101E: Error saving configuration, changes not saved due to exception Exc. Value is:

Explanation: This is an error message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response:This message is self-explanatory.

UDIN9102E: Error saving configuration, changes not saved due to exception Exc. Value is:

Explanation: This is an error message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response:This message is self-explanatory.

UDIN9103E: Error saving configuration, changes not saved due to exception Exc. Value is:

Explanation: This is an error message issued by the UDDI setup script appserverremoveuddi.jacl.

User Response:This message is self-explanatory.

UDLC (Web Services UDDI) messages

There are no messages issued by this component.

UDPR (Web Services UDDI) messages

There are no messages issued by this component.

UDRS (Web Services UDDI) messages

UDRS0001E: Exception "<exception>" occurred while attempting to get UDDI Message Logger.

Explanation: This message is issued to stderr when an attempt to get the UDDI Message Logger fails with the indicated exception. Since the attempt to get the message logger failed, the message cannot be logged. No messages can be logged by this instance of the IBM WebSphere UDDI Registry.

User Response: Restart the UDDI registry. If the error persists, examine the WebSphere logs for information on its cause. If the problem cannot be resolved, then please contact the IBM Customer Service Center.

UDRS0002E: Exception "<exception>" occurred while attempting to get UDDI Trace Logger for "<component>".

Explanation: This message is logged when an attempt to get the UDDI Trace Logger for the specified component (or package) fails with the indicated exception. No trace entries can be logged for this component or package of the IBM WebSphere UDDI Registry.

User Response: Restart the UDDI registry. If the error persists, examine the WebSphere logs for information on its cause. If the problem cannot be resolved, then please contact the IBM Customer Service Center.

UDSC (Web Services UDDI) messages

There are no messages issued by this component.

UDSP (Web Services UDDI) messages

UDSP0001E: ParserPool found empty whilst attempting to process request. Request unsatisfied

Explanation: A SOAP request was received, but was unable to be dealt with, as there were no free Parsers within the ParserPool.

User Response: Consider increasing the number of Parsers within the ParserPool by modifying the Init Parameter on the SOAP servlets.

UDSP0002E: Error locating schemas required for UDDI processing. SOAP Servlets unworkable.

Explanation: The SOAP servlet was unable to locate the schemas it requires in order to process SOAP requests. Without these, the servlet cannot process SOAP requests.

User Response: Check installation of UDDI was performed correctly. If the error persists, examine the WebSphere logs for information on its cause. If the problem cannot be resolved, then please contact the IBM Customer Service Center.

UDSP0003W: Servlet unable to locate init parameter 'defaultPoolSize'. Using internal defaults.

Explanation: The SOAP servlet was unable to locate the init parameter which sets the default size of the ParserPool. It will fall back to an internal default.

User Response: If this message occurred after attempting to make changes to the defaultPoolSize init parameter, ensure the changes were correct. If this message has appeared after installed, ensure installation was performed correctly.

**UDSP0004W: Servlet unable to understand init parameter 'defaultPoolSize'.
Using internal defaults.**

Explanation: The SOAP servlet was unable to parse the init parameter which sets the default size of the ParserPool. It will fall back to an internal default.

User Response: If this message occurred after attempting to make changes to the defaultPoolSize init parameter, ensure the changes were correct. If this message has appeared after installation, ensure installation was performed correctly.

UDSP0005E: Error occurred during parser creation.

Explanation: An unspecified error occurred during the creation of a SOAP parser

User Response: Restart the UDDI registry. If the error persists, examine the WebSphere logs for information on its cause. If the problem cannot be resolved, then please contact the IBM Customer Service Center.

UDSP0006E: Internal configuration error.

Explanation: This error may occur if there was a failure creating a Parser, with accompanying message UDSP0005. It may also occur if there was a problem acquiring the Persistence layer.

User Response: Restart the UDDI registry. If the error persists, examine the WebSphere logs for information on its cause. If the problem cannot be resolved, then please contact the IBM Customer Service Center.

UDSP0007E: Error during servlet acquisition of persistence layer.

Explanation: The SOAP servlet was unable to acquire the persistence layer required for it to communicate with the UDDI datasource

User Response: Restart the UDDI registry. If the error persists, examine the WebSphere logs for information on its cause. If the problem cannot be resolved, then please contact the IBM Customer Service Center.

UDSP0008E: Error during servlet release of persistence layer.

Explanation: The persistence layer reported a problem when the SOAP servlet attempted to release it.

User Response: Restart the UDDI registry. If the error persists, examine the WebSphere logs for information on its cause. If the problem cannot be resolved, then please contact the IBM Customer Service Center.

UDSP0009E: Error during sending of response to client.

Explanation: An error occurred when sending a SOAP response message back to a client. The client may not have received the response

User Response: This error is recorded to enable logging of failed responses to clients. The error may be the fault of the client disconnecting before the reply could be sent, or may indicate a network problem. Examine the WebSphere logs for more information on its cause.

UDUC (Web Services UDDI) messages

UDUC0001I: IBM WebSphere UDDI Registry user console starting initialization.

Explanation: The user console control servlet is starting.

User Response: None.

UDUC0002I: IBM WebSphere UDDI Registry user console finished initialization.

Explanation: The user console control servlet has completed startup.

User Response: None.

UDUC0003I: Reading init parameters.

Explanation: The user console control servlet has started reading external parameters in its init method

User Response: None.

UDUC0004I: Finished reading init parameters.

Explanation: The user console control servlet has finished reading external parameters in its init method. This message indicates the user console is ready to accept client requests.

User Response: None.

UDUC0005E: A serious error has occurred. Error message: <Message> error: <Throwable>. More information: <Additional information>.

Explanation: This error message indicates an unexpected error has occurred. The <Message> describes the error that has occurred and the <Throwable> is the type of error that was caught. <Additional information> may provide further information, if available.

User Response: A trace of the gui component is recommended. Contact IBM support with this information.

UDUC0006E: A persistence error has occurred. Error message: <Message> error: <Throwable>. More information: <Additional information>.

Explanation: An error occurred while performing a database operation. The <Message> describes the error that occurred and the <Throwable> is the type of error that was caught. <Additional information> may provide further information, if available.

User Response: Check database connections and state. Please provide IBM support with a trace, including the gui and persistence components.

UDUC0007E: A User mismatch error has occurred. Error message: <Message> error: <Throwable>. More information: <Additional information>.

Explanation: The user id provided does not match the user id required or expected whilst performing an operation that requires authentication. The <Message> describes the error that occurred and the <Throwable> is the type of error that was caught. <Additional information> may provide further information, if available.

User Response: Check the user has authority for the operation being requested. If necessary, contact IBM support detailing the actions taken to recreate the problem.

UDUC0008E: An invalid key was passed. Error message: <Message> error: <Throwable>. More information: <Additional information>.

Explanation: The requested operation is trying to retrieve information about an entity with a key that is invalid. This may occur if the entity has been deleted by another session. The <Message> describes the error that occurred and the <Throwable> is the type of error that was caught. <Additional information> may provide further information, if available.

User Response: Ask the client to close existing sessions and attempt the operation in a new browser session. If the problem persists, please provide IBM support with a trace of the gui and api components.

UDUC0009E: An invalid value was supplied. Error message: <Message> error: <Throwable>. More information: <Additional information>

Explanation: An invalid value was passed to an API call. The >Message> describes the error that occurred and the <Throwable> is the type of error that was caught. <Additional information> may provide further information, if available.

User Response: Contact IBM support with a trace of the gui and api components.

UDUC0010E: Failed to introspect ActionForm properties. Exception:

<Exception>.

Explanation: String properties of a form object could not be introspected which means that the form contents cannot be checked for invalid characters.

User Response: Please contact IBM support with details of the Exception and a trace of the gui component.

UDUC0011E: Failed to invoke reflected methods in ActionForm. Exception:

<Exception>.

Explanation: A form object's declared public method for setting or getting a String value could not be invoked. This method is required to check for invalid characters.

User Response: Please contact IBM support with details of the Exception and a trace of the gui component.

UDUC0012E: User console initialization failed to connect to UDDI database.

Exception: <Exception>.

Explanation: During user console initialization, connection to the database failed, and threw the exception specified.

User Response: Check the connection to the UDDI database. The included exception message may yield some clues to help you resolve the problem. If unresolved, please contact IBM support with a trace of the gui component during startup.

UDUC0013E: User console initialization failed to initialize tModels. Exception:

<Exception>.

Explanation: Indicates that an error has occurred during initialization of ActionServlet, specifically when reading tModels (invoking init method in class TModelNames).

User Response: Check the state of the UDDI database. Visually inspect the TMODEL table and confirm it is populated with valid data. The included exception message may yield some clues to help you resolve the problem. If unresolved, please contact IBM support with a trace of the gui component during startup.

UDUC0014E: User console initialization failed to initialize taxonomies.

Exception: <Exception>.

Explanation: Indicates that an error has occurred during initialization of ActionServlet, specifically when reading taxonomy data (invoking init method of CategoryTaxonomyTree).

User Response: Check the state of the UDDI database. The included exception message may yield some clues to help you resolve the problem. If unresolved, please contact IBM support with a trace of the gui component during startup.

UDUT UDDI Utility Tools messages

UDUT0001I: Usage: java -jar UDDIUtilityTools.jar *'function'* [options]

function:

-promote <i>entity source</i>	Promote entities between registries
-export <i>entity source</i>	Extract entities from registry to XML
-delete <i>entity source</i>	Delete entities from registry
-import	Create entities from XML to registry

where *entity source* is one of:

-tmodel|-business|-service|-binding *key* **Specify single entity type and key**

-keysFile | -f filename Specify file containing entity types and keys

options:

-properties filename Specify path to configuration file
-overwrite | -o Overwrite an entity if it already exists
-log | -v Output verbose messages
-definitionFile filename Specify path to UDDI entity definition file
-importReferenced Import entities referenced by source entities

The following options override property settings in configuration file:

-overwrite
-log
-definitionFile
-importReferenced

Example: `java -jar UDDIUtilityTools.jar -promote -keysFile C:/uddikeys.txt`

Explanation: This is the usage message displayed at the command line when the user has entered an invalid combination of arguments or options.

User Response: Enter the command according to the usage message..

UDUT0002I: *** Starting UDDI Utility Tools**

Explanation: This message is used as a marker in the message log file to indicate tool start points.

User Response: None.

UDUT0003I: Promoting entityType<entity type> key<entity key>...

Explanation: Indicates which entity type (business, tModel and so on) is being promoted, and it's key value.

User Response: None.

UDUT0004I: Bad entityType: received<incorrect entity type>, expected <tModel|business|service|binding>

Explanation: The user entered an incorrect entity type.

User Response: Use an entity type of tModel, business, service or binding.

UDUT0005I: Promotion successful.

Explanation: Indicates the promote function completed successfully.

User Response: None.

UDUT0006I: Import successful.

Explanation: Indicates the import function completed successfully.

User Response: None.

UDUT0007I: Export successful.

Explanation: Indicates the export function completed successfully.

User Response: None.

UDUT0008I: Delete successful.

Explanation: Indicates the delete function completed successfully.

User Response: None.

UDUT0009I: Exporting entities ...

Explanation: Indicates the export function has started.

User Response: None.

UDUT0010I: Exporting business, businessKey[<business key].

Explanation: Indicates that the businessEntity with the specified key is being exported.

User Response: None.

- UDUT0011I: Exporting service, serviceKey[<service key>].**
Explanation: Indicates that the businessService with the specified key is being exported.
User Response: None.
- UDUT0012I: Exporting binding, bindingKey[<binding key>].**
Explanation: Indicates that the bindingTemplate with the specified key is being exported.
User Response: None.
- UDUT0013I: Exporting tModel, tModelKey[<tModel key>].**
Explanation: Indicates that the tModel with the specified key is being exported.
User Response: None.
- UDUT0014I: Exporting referenced tModel, tModelKey[<tModel key>].**
Explanation: Indicates that the referenced tModel with the specified key is being exported.
User Response: None.
- UDUT0015I: Exported <entity count> entities.**
Explanation: Indicates that the export function completed, and shows the number of entities exported.
User Response: None.
- UDUT0016I: Importing entities ...**
Explanation: Indicates the import function has started.
User Response: None.
- UDUT0017I: Importing business, businessKey[<business key>].**
Explanation: Indicates that the businessEntity with the specified key is being imported.
User Response: None.
- UDUT0018I: Importing service, serviceKey[<service key>].**
Explanation: Indicates that the businessService with the specified key is being imported.
User Response: None.
- UDUT0019I: Importing binding, bindingKey[<binding key>].**
Explanation: Indicates that the bindingTemplate with the specified key is being imported.
User Response: None.
- UDUT0020I: Importing tModel, tModelKey[<tModel key>].**
Explanation: Indicates that the tModel with the specified key is being imported..
User Response: None.
- UDUT0021I: Importing referenced tModel, tModelKey[<tModel key>].**
Explanation: Indicates that the referenced tModel with the specified key is being imported.
User Response: None.
- UDUT0022I: Imported <entity count> entities.**
Explanation: Indicates that the import function completed, and shows the number of entities imported.
User Response: None.
- UDUT0023I: Deleting entities ...**
Explanation: Indicates the delete function has started.

- User Response:** None.
- UDUT0024I: Deleting business, businessKey[<business key>].**
Explanation: Indicates that the businessEntity with the specified key is being deleted.
- User Response:** None.
- UDUT0025I: Deleting service, serviceKey[<service key>].**
Explanation: Indicates that the businessService with the specified key is being deleted.
- User Response:** None.
- UDUT0026I: Deleting binding, bindingKey[<binding key>].**
Explanation: Indicates that the bindingTemplate with the specified key is being deleted.
- User Response:** None.
- UDUT0027I: Deleting tModel, tModelKey[<tModel key>].**
Explanation: Indicates that the tModel with the specified key is being deleted.
- User Response:** None.
- UDUT0028I: Deleted <entity count> entities.**
Explanation: Indicates that the delete function completed, and shows the number of entities deleted.
- User Response:** None.
- UDUT0029I: Serializing ...**
Explanation: Indicates that generation of the Entity Definition File has started.
- User Response:** None.
- UDUT0030I: Serialized entities.**
Explanation: Indicates that generation of the Entity Definition File completed successfully.
- User Response:** None.
- UDUT0031I: Deserializing ...**
Explanation: Indicates that reading of the Entity Definition File and creation of UDDI entities has started.
- User Response:** None.
- UDUT0032I: Deserialized entities.**
Explanation: Indicates that reading of the Entity Definition File and creation of UDDI entities completed successfully.
- User Response:** None.
- UDUT0033I: Function '<function>' completed successfully.**
Explanation: Indicates the requested function completed successfully.
- User Response:** None.
- UDUT0034W: Function '<function>' did not complete successfully. See message log for further information.**
Explanation: Indicates the requested function did not complete successfully.
- User Response:** The message log may yield further information if the verbose option is on. Check the configuration properties file setting are correct. If that does not identify the problem, try running with trace logging enabled. If that does not yield a solution, contact your IBM support center.

- UDUT0035W: Parser error: {0}**
Explanation: The XML parser reports a warning about the content of the Entity Definition File.
User Response: Based on the context of the warning message, check the validity of the Entity Definition File.
- UDUT0036E: Parser error {0}**
Explanation: The XML parser reports an error about the content of the Entity Definition File.
User Response: Based on the context of the error message, check the validity of the Entity Definition File.
- UDUT0037E: Unrecognized parser feature: <feature>**
Explanation: A parser feature set by the UDDI Utility Tools is not recognized by the parser.
User Response: Check you are using the correct type and level of XML parser. If correct, contact your IBM support center.
- UDUT0038E: Unsupported parser feature: <feature>**
Explanation: A parser feature set by the UDDI Utility Tools is not supported by the parser.
User Response: Check you are using the correct type and level of XML parser. If correct, contact your IBM support center.
- UDUT0039E: Unrecognized parser property: <property>, value: <value>**
Explanation: A parser property set by the UDDI Utility Tools is not recognized by the parser.
User Response: Check you are using the correct type and level of XML parser. If correct, contact your IBM support center.
- UDUT0040E: Unsupported parser property: <property>, value: <value>**
Explanation: A parser property set by the UDDI Utility Tools is not supported by the parser.
User Response: Check you are using the correct type and level of XML parser. If correct, contact your IBM support center.
- UDUT0041I: <message>**
Explanation: This is a placeholder message used during development only.
User Response: None.
- UDUT0042E: Unable to find the configuration file: <filepath>**
Explanation: UDDI Utility Tools cannot locate the specified configuration file.
User Response: UDDI Utility Tools looks for a default configuration properties with the file name 'UDDIUtilityTools.properties' in the current directory. Check that the configuration file has this name, or that the argument value supplied with the '-properties' option is pointing at a file that exists.
- UDUT0043E: An Exception occurred trying to read the configuration file.**
Explanation: The configuration file could not be read.
User Response: Check the file path points to a valid file and that the file does not have the 'hidden' attribute set.
- UDUT0044W: Configuration file is missing the '<property name>' property.**
Explanation: A required property is missing from the configuration file.
User Response: Add the missing property name and value to the configuration file. Check that the property name is not misspelled.

UDUT0045W: Property: '<property name>' has value '<property value>'. It must be either 'true' or 'false'.

Explanation: A value was given to a property other than 'true' or 'false'.

User Response: Set the property value to 'true' or 'false'.

UDUT0046W: Property: '<property name>' has value '<property value>'. It must be an integer value.

Explanation: A value was given to a property other than an integer value.

User Response: Set the property value to an integer value.

UDUT0047E: Unable to find the keyFile file: <keys file path>

Explanation: The keys file could not be located at the specified path.

User Response: Check the file name and path and correct and that the file exists.

UDUT0048E: Unable to read the keyFile file: <keys file path>

Explanation: The keys file could not be read due to an IO error.

User Response: Check the file's hidden attribute is not set.

UDUT0049E: Unable to write to entity definition file: <entity definition file path> **Explanation:** During initialization, the Entity Definition File could not be written to.acl.

User Response: Check the file's read only attribute is not set.

UDUT0050E: Unable to find UDDI Entity definition file: <entity definition file path>

Explanation: The Entity Definition File could not be found at the specified file path.

User Response: Check the file path is correct and that the file exists.

UDUT0051E: Unable to read UDDI Entity definition file: <entity definition file path>

Explanation: The Entity Definition File could not be read due to an IO error.

User Response: Check the file's hidden attribute is not set.

UDUT0052E: Unable to close the message file: <file path>

Explanation: The attempt to close the message file failed.

User Response: The disk might be full. If so, clear some space or direct log output to a different disk.

UDUT0053E: Unable to close the trace file: <file path>

Explanation: The attempt to close the trace log file failed.

User Response: The disk might be full. If so, clear some space or direct log output to a different disk.

UDUT0054E: The logger was unable to find the file: <file path>

Explanation: The UDDI Utility Tools logger could not find the specified file.

User Response: None.

UDUT0055E: ERROR OCCURRED ...

Explanation: General purpose error message used in development only.

User Response: None.

UDUT0056E: Exception:

Explanation: General purpose message prefix used for reporting exceptions.

User Response: None.

- UDUT0057W: Only one function may be specified on the command line.**
Explanation: Multiple function commands were entered on the command line.
User Response: Specify one function in accordance with the usage message.
- UDUT0058W: No function was specified.**
Explanation: UDDI Utility Tools was invoked with no function specified.
User Response: Specify one function in accordance with the usage message.
- UDUT0059W: The function: <function> was not recognized.**
Explanation: The function value did not match any of the allowed functions.
User Response: Specify one function in accordance with the usage message.
- UDUT0060W: The argument '<argument>' was not recognized.**
Explanation: The argument value does not match any of the allowed arguments.
User Response: Specify arguments in accordance with the usage message.
- UDUT0061W: There was a missing value for <argument> argument.**
Explanation: An expected value for the specified argument was not supplied.
User Response: Specify a value for the argument in accordance with the usage message.
- UDUT0062W: Unexpected argument: <argument> (entity key file is already specified).**
Explanation: The entity type argument cannot be specified if the keysFile argument is already specified.
User Response: Specify arguments in accordance with the usage message.
- UDUT0063W: Unexpected argument: <argument> (entity key is already specified).**
Explanation: The keysFile argument cannot be specified if an entity type argument and key value has already been specified.
User Response: Specify arguments in accordance with the usage message.
- UDUT0064W: Argument: <argument> cannot be specified more than once.**
Explanation: An argument was specified twice in the same command.
User Response: Specify arguments in accordance with the usage message.
- UDUT0065E: No entity keys were specified.**
Explanation: A keys file or an entity type and key value must be specified for functions using keys.
User Response: Specify arguments in accordance with the usage message.
- UDUT0066E: Could not load Database driver: dbDriver<database driver>.**
Explanation: The specified database driver could not be loaded.
User Response: Check the database driver value in the configuration file is valid, and the driver's class is present in the classpath property.
- UDUT0067E: Could not create database connection: dbUrl<database URL. dbUser:<database userid>, (dbPasswd not shown).**
Explanation: A connection could not be established with the database at the specified URL with the specified userid.
User Response: Check the database URL, userid and password values are correct in the configuration file, and that the database manager is running.

UDUT0068E: Could not close the database connection.

Explanation: An attempt to close the database connection failed.

User Response: If the problem persists, contact your IBM support center.

UDUT0069E: Could not create minimal entity to tModel.

Explanation: The minimal data necessary for a valid tModel could not be inserted in the target UDDI registry database.

User Response: Check the database URL, userid and password values are correct in the configuration file, and that the database manager is running.

UDUT0070E: Could not create minimal entity for Service.

Explanation: The minimal data necessary for a valid businessService could not be inserted in the target UDDI registry database.

User Response: Check the database URL, userid and password values are correct in the configuration file, and that the database manager is running.

UDUT0071E: Could not create minimal entity for Business.

Explanation: The minimal data necessary for a valid businessEntity could not be inserted in the target UDDI registry database.

User Response: Check the database URL, userid and password values are correct in the configuration file, and that the database manager is running.

UDUT0072E: Could not create minimal entity for Binding.

Explanation: The minimal data necessary for a valid bindingTemplate could not be inserted in the target UDDI registry database.

User Response: Check the database URL, userid and password values are correct in the configuration file, and that the database manager is running.

UDUT0073E: There was an error while trying to create an XML Document.

Explanation: An attempt to create the Entity Definition File failed.

User Response: Check the file path specified in the configuration file for the Entity Definition File is valid and is not set to read only.

UDUT0074E: There was an error parsing the entity definition file.

Explanation: An unspecified error occurred when parsing the Entity Definition File.

User Response: Check the entity definition file content is valid according to the UDDI Utility Tools schema file, promoter.xsd.

UDUT0075E: One or more errors occurred while parsing the entity definition file. See message log for details.

Explanation: Errors occurred when parsing the Entity Definition File.

User Response: Check the entity definition file content is valid according to the UDDI Utility Tools schema file, promoter.xsd.

UDUT0076W: One or more warnings were raised while parsing the entity definition file. See message log for details.

Explanation: Warnings occurred when parsing the Entity Definition File.

User Response: Check the entity definition file content is valid according to the UDDI Utility Tools schema file, promoter.xsd.

UDUT0077E: An error occurred trying to discover the publisher.

Explanation: EEE.

User Response: None.

UDUT0078E: Unable to obtain authinfo.

Explanation: EEE.

User Response: None.

- UDUT0079E: The inquiryURL is malformed: <inquiry URL>.**
Explanation: The inquiry URL specified in the configuration file is not valid.
User Response: Correct the value for the inquiry URLs (fromInquiryURL and toInquiryURL) in the configuration file.
- UDUT0080E: The publisherURL is malformed: <publish URL>**
Explanation: The publish URL specified in the configuration file is not valid.
User Response: Correct the value for the publish URL (toPublishURL) in the configuration file.
- UDUT0081E: Could not get tModel detail for tModelKey[<tModel key>].**
Explanation: The get tModel operation failed on the source registry.
User Response: Check the key exists in the source registry.
- UDUT0082E: Could not get service detail for serviceKey[<service key>].**
Explanation: The get service operation failed on the source registry.
User Response: Check the key exists in the source registry.
- UDUT0083E: Could not get business detail for businessKey[<business key>].**
Explanation: The get business operation failed on the source registry.
User Response: Check the key exists in the source registry.
- UDUT0084E: Could not get binding detail for bindingKey[<binding key>].**
Explanation: The get binding operation failed on the source registry.
User Response: check the key exists in the source registry.
- UDUT0085E: Could not save tModel for tModelKey[<tModel key>].**
Explanation: The publish operation failed at the target registry.
User Response: Check the tModel is not referencing another entity (such as a tModel) that is not present in the target registry. This may occur if the 'importReferenced' property is set to false. Specify referenced tModels in the referencedtModels section of the Entity Definition File and set 'importReferenced' property in the configuration file to 'true'.
- UDUT0086E: Could not save business for businessKey[<business key>].**
Explanation: The publish operation failed at the target registry.
User Response: Check the businessEntity is not referencing another entity (such as a tModel) that is not present in the target registry. This may occur if the 'importReferenced' property is set to false. Specify referenced tModels in the referencedtModels section of the Entity Definition File and set 'importReferenced' property in the configuration file to 'true'.
- UDUT0087E: Could not save service for parent businessKey[<business key>].**
Explanation: The publish operation failed at the target registry.
User Response: Check the businessEntity specified as the parent of the businessService exists in the target registry.
- UDUT0088E: Could not save binding for parent serviceKey[<service key>].**
Explanation: The publish operation failed at the target registry.
User Response: Check the businessService specified as the parent of the bindingTemplate exists in the target directory.
- UDUT0089W: Did not save service for serviceKey[<service key>].**
Explanation: The parent business for the specified businessService does not exist.
User Response: Check the key value for the parent entity is correct in the Entity Definition File, and that the entity exists in the target registry.

UDUT0090W: Did not save binding for bindingKey[<binding key>].

Explanation: The parent service for the specified bindingTemplate does not exist.

User Response: Check the key value for the parent entity is correct in the Entity Definition File, and that the entity exists in the target registry.

UDUU (Web Services UDDI) messages

There are no messages issued by this component.

Running the UDDI Samples

The UDDI samples, and documentation on how to use them, are available through the Web Services UDDI samples link on the Samples Central page of the IBM WebSphere Developer Domain Web site.

Installation Verification Program (IVP)

There are some samples available on the WSDD web site (at <http://www7b.software.ibm.com/wsdd/library/samples/AppServer.html>) that are intended to provide an optional Installation Verification test, or IVP, for the UDDI Registry component.

This topic describes how to run these installation verification programs (IVPs) to verify that the IBM UDDI Registry has been installed correctly.

There are two IVP SOAP samples called SOAPSsampleIVPa and SOAPSsampleIVPb. They are intended to verify the successful installation of the product, and should be used in conjunction with the UDDI Users Console (GUI). SOAPSsampleIVPa saves some data to the registry which you can then find using the GUI. Finally you can delete the data by running SOAPSsampleIVPb.

The IVP samples are installed into the same target directory as the other SOAP samples and they use the same XML files as the basic Java SOAP samples.

SOAPSsampleIVPa saves three businesses, six services (2 per business) and three tModels. The data structures are very basic and consist only of a name. The keys returned by the save_* UDDI API calls are then written to a file, SOAPSsampleIVPa.out. SOAPSsampleIVPb then reads in these keys from the file in order to delete the saved data from the UDDI registry.

Note: Each time you run SOAPSsampleIVPa, it overwrites the output file SOAPSsampleIVPa.out so, if you wish to use SOAPSsampleIVPb to delete the data, you must run this before you next run SOAPSsampleIVPa.

To run the IVPs, complete the following steps on the same system as the UDDI Registry:

1. Ensure that DB2 and the WebSphere Admin Server are started.
2. Start the WebSphere Administrator's Console and ensure the default server is started and the UDDI Registry Application is started.
3. For SOAP samples to work, you need to ensure that the Client Developer Kit for Java is either the one shipped with IBM WebSphere Application Server or a later IBM Developer Kit for Java:
 - For Windows - ensure that <WebSphere-install-dir>\java\bin is present in the PATH statement before any other Developer Kits for Java

- For Unix Platforms - ensure that <WebSphere-install-dir>/java/bin is present in the PATH statement before any other Developer Kits for Java

Note: You **must** use the IBM WebSphere supplied Developer Kit for Java or a later level of the IBM Developer Kit for Java.

For Windows, the default system path can be set via **Control Panel ...-> Settings ...-> System ...-> Advanced Properties ...-> Environment Variables**

Alternatively, this can be accomplished just for the shell where you plan to run the samples by modifying the path within the shell:

- For Windows - set path=<WebSphere-install-dir>\java\bin;%path%
- For Unix Platforms - export PATH=<WebSphere-install-dir>/java/bin:\$PATH

4. Copy the samples and *.xml files to a directory, and compile and run them there (see next bullets)
5. Compile both SOAPSsampleIVPa and SOAPSsampleIVPb by typing:

```
'javac SOAPSsampleIVPa.java'
```

and

```
'javac SOAPSsampleIVPb'
```

6. Run SOAPSsampleIVPa by typing 'java SOAPSsampleIVPa'. This should publish a number of businesses and services and technical models into the registry.
7. Start your Web browser on the same system as the UDDI Registry.
8. To display the UDDI GUI home page, type one of the following URLs:
 - If you have WebSphere security disabled: http://localhost:9080/uddigui
 - If you have WebSphere security enabled: https://localhost:9433/uddigui
9. On the find page, complete the following steps:
 - a. Select the business radio button
 - b. In the data entry field, type % (percent is the wild card symbol)
 - c. Click **Find**

You should get a results page returned with three businesses (mybusiness1, mybusiness2, and mybusiness3). This demonstrates that the API and the UDDI Console are working correctly.

10. To see the services that are available for a business, click the "Show Services" option next to the business.
11. To delete all of the IVP data, run SOAPSsampleIVPb (from the command prompt as before - by typing 'java SOAPSsampleIVPb')
12. On the find page, complete the following steps:
 - a. Select the business radio button
 - b. In the data entry field, type % (percent is the wild card symbol)
 - c. Click **Find**

You should get an empty results page returned.

Reporting problems with the IBM WebSphere UDDI Registry

If you report a problem with the IBM WebSphere UDDI Registry component to IBM, please supply the following information:

1. A detailed description of the problem.
2. The build date and time of the version you are using. This can be obtained as follows:

- In the installedApps subdirectory of the WebSphere install location, you will find a subdirectory called UDDI_Registry.<nodename>.<servername>.ear, where <nodename> is the name of the node into which the UDDI Registry application is installed, and <servername> is the name of the server. Within that subdirectory, you will find a file called version.txt. Please include the contents of this file as part of your information.
 - If the UDDI Registry has been started with tracing enabled for the UDDI component, then you should find a trace entry in the WebSphere trace log which includes the strings " getUDDIMessageLogger" and "UDDI Build : " followed by the build date and time, and the build system. Please also include this information.
3. Any relevant log files and trace files.
 - If the problem occurred while setting up and installing the UDDI Registry application using one of the setup scripts, setupuddi.jacl or appserversetupuddi.jacl, then please supply the log output from running the script. (If you had not chosen to redirect the output from the script file to a log file, then please rerun the script, this time redirecting the output as described in the section 'Installing and Setting up a UDDI Registry'.) The log file will be in the directory from which you ran the setup script.
 - If the problem occurred while removing the UDDI Registry application using one of the remove scripts, removeuddi.jacl or appserverremoveuddi.jacl, then please supply the log output from running the script. (If you had not chosen to redirect the output from the script file to a log file, then please rerun the script, this time redirecting the output as described in the section 'Removing the UDDI Registry from a deployment manager cell' or 'Removing the UDDI Registry application from a single appserver'.) The log file will be in the directory from which you ran the remove script.
 - If the problem occurred while creating the UDDI Registry database using the UDDI DB2 Setup Wizard, then please supply the log file UDDIloadDB.log, which will be in the directory from which the wizard was run.
 - If the problem occurred while running the UDDI Registry, please enable UDDI tracing (if not already enabled) and supply the trace log from the logs directory of the application server on which the UDDI Registry was running. Please refer to the section on 'Turning on UDDI Trace' for details on how to enable UDDI tracing.
 4. If appropriate, any application code that you are using and the output produced by the application code.

In addition to the above, it might be useful to run the WebSphere collector tool and send the resulting jar file(s) (two files if run from Base AppServer AND DeploymentManager) to IBM. Instructions on running the collector tool can be found at [Running the collector tool](#)

Feedback

See the section on "Obtaining help from IBM" elsewhere in this InfoCenter for details on seeking assistance.

Chapter 9. Class loading

Class loaders affect the packaging of applications and the run-time behavior of packaged applications deployed on application servers.

1. Read about class loaders. The article "“Class loading: Resources for learning” on page 423” points to additional sources.
2. If necessary, migrate class-loader Module Visibility Mode settings for Version 4.0.x applications to Version 5.0 application or WAR class-loader policies.
3. When assembling an enterprise application resource (EAR) file that has EJB modules, set the classpath for the class loader to use during packaging.
4. If an application module uses a resource, create a resource provider that specifies the directory name of the resource drivers. Do not specify the resource JAR file names. All JAR files in the specified directory will be added into the classpath of the WebSphere Application Server extensions class loader.
5. Configure class loaders of an application server for the run-time environment.

- a. Click **Servers > Application Servers > *server_name*** and, on the settings page for an application server, set the application class-loader policy and application class-loader mode.

The application class-loader policy controls the isolation of applications running in the system. When set to **SINGLE**, applications are not isolated; a single application class loader is used to contain all EJB modules, dependency JAR files, and shared libraries in the system. When set to **MULTIPLE**, applications are isolated from each other; each application receives its own class loader to load that application’s EJB modules, dependency JAR files, and shared libraries.

The application class-loader mode specifies the class-loader mode when the application class-loader policy is **SINGLE**. **PARENT_FIRST** causes the class loader to first delegate the loading of classes to its parent class loader before attempting to load the class from its local classpath. **PARENT_LAST** causes the class loader to first attempt to load classes from its local classpath before delegating the class loading to its parent. This allows an application class loader to override and provide its own version of a class that exists in the parent class loader.

- b. On the settings page for an application server, click **Classloader**. On the Classloader page, click **New**.
 - c. On the settings page for a class loader, specify the class-loader mode. **PARENT_FIRST** causes the class loader to delegate the loading of classes to its parent class loader before attempting to load the class from its local classpath. **PARENT_LAST** causes the class loader to attempt to load classes from its local classpath before delegating the class loading to its parent. Then, click **OK**.
 - d. On the settings page for a class loader, click **Libraries**. From the Library Ref page, click **Add**. On the settings page for a library reference, specify variables for the library reference as needed and click **OK**. Repeat the previous step until you define a library reference instance for each library file that your application needs. To define a library reference, you must first define one or more shared libraries.
6. When configuring an installed enterprise application for deployment in the run-time environment, set the class-loader mode and the WAR class-loader policy.

7. When configuring an installed Web module for deployment in the run-time environment, set the class-loader mode.

Class loaders

Class loaders are part of the Java virtual machine (JVM) code and are responsible for finding and loading class files. Class loaders affect the packaging of applications and the run-time behavior of packaged applications deployed on application servers.

The run-time environment of WebSphere Application Server uses the following class loaders to find and load new classes for an application in the following order:

1. The **bootstrap, extensions, and CLASSPATH class loaders** created by the JVM.

The bootstrap class loader uses the boot classpath (typically classes in `jre/lib`) to find and load classes. The extensions class loader uses the system property `java.ext.dirs` (typically `jre/lib/ext`) to find and load classes. The CLASSPATH class loader uses the CLASSPATH environment variable to find and load classes.

The CLASSPATH class loader contains the J2EE APIs of the WebSphere Application Server product (inside `j2ee.jar`). Because the J2EE APIs are in this class loader, you can add libraries that depend on J2EE APIs to the classpath system property to extend a server's classpath. However, a preferred method of extending a server's classpath is to add a shared library.

2. A **WebSphere-specific extensions class loader**.

The WebSphere extensions class loader loads the WebSphere run-time and J2EE classes that are required at run time. The extensions class loader uses a `ws.ext.dirs` system property to determine the path used to load classes. Each directory in the `ws.ext.dirs` classpath and every JAR file or ZIP file in these directories is added to the classpath used by this class loader.

The WebSphere extensions class loader also loads resource provider classes into a server if an application module installed on the server refers to a resource that is associated with the provider and if the provider specifies the directory name of the resource drivers.

3. One or more **application module class loaders** that load elements of enterprise applications running in the server.

The application elements can be Web modules, EJB modules, resource adapters, and dependency JAR files. Application class loaders follow J2EE class-loading rules to load classes and JAR files from an enterprise application. The WebSphere run time enables you to associate a shared library classpath with an application.

Each class loader is a child of the class loader above it. That is, the application module class loaders are children of the WebSphere-specific extensions class loader, which is a child of the CLASSPATH Java class loader. Whenever a class needs to be loaded, the class loader usually delegates the request to its parent class loader. If none of the parent class loaders can find the class, the original class loader attempts to load the class. Requests can only go to a parent class loader; they cannot go to a child class loader. If the WebSphere class loader is requested to find a class in a J2EE module, it cannot go to the application module class loader to find that class and a `ClassNotFoundException` occurs. Once a class is loaded by a class loader, any new classes that it tries to load reuse the same class loader or go up the precedence list until the class is found.

Class-loader isolation policies

The number and function of the application module class loaders depends on the class-loader policies specified in the server configuration. Class loaders provide multiple options for isolating applications and modules to enable different application packaging schemes to run on an application server.

Two class-loader policies control the isolation of applications and modules:

Application class-loader policy

Application class loaders consist of EJB modules, dependency JAR files, resource adapters, and shared libraries. Depending on the application class-loader policy, an application class loader can be shared by multiple applications (SINGLE) or unique for each application (MULTIPLE). The application class-loader policy controls the isolation of applications running in the system. When set to SINGLE, applications are not isolated. When set to MULTIPLE, applications are isolated from each other.

WAR class-loader policy

By default, Web module class loaders load the contents of the WEB-INF/classes and WEB-INF/lib directories. The application class loader is the parent of the Web module class loader. You can change the default behavior by changing the application's WAR class-loader policy.

The WAR class-loader policy controls the isolation of Web modules. If this policy is set to APPLICATION, then the Web module contents also are loaded by the application class loader (in addition to the EJB files, RAR files, dependency JAR files, and shared libraries). If the policy is set to MODULE, then each web module receives its own class loader whose parent is the application class loader.

Note: WebSphere server class loaders never load application client modules.

For each application server in the system, you can set the application class-loader policy to SINGLE or MULTIPLE. When the application class-loader policy is set to SINGLE, then a single application class loader loads all EJB modules, dependency JAR files, and shared libraries in the system. When the application class-loader policy is set to MULTIPLE, then each application receives its own class loader used for loading that application's EJB modules, dependency JAR files, and shared libraries.

This application class loader can load each application's Web modules if that WAR module's class-loader policy is also set to APPLICATION. If the WAR module's class-loader policy is set to APPLICATION, then the application's loader loads the WAR module's classes. If the WAR class-loader policy is set to MODULE, then each WAR module receives its own class loader.

The following example shows that when the application class-loader policy is set to SINGLE, a single application class loader loads all EJB modules, dependency JAR files, and shared libraries of all applications on the server. The single application class loader can also load Web modules if an application has its WAR class-loader policy set to APPLICATION. Applications having a WAR class-loader policy set to MODULE use a separate class loader for Web modules.

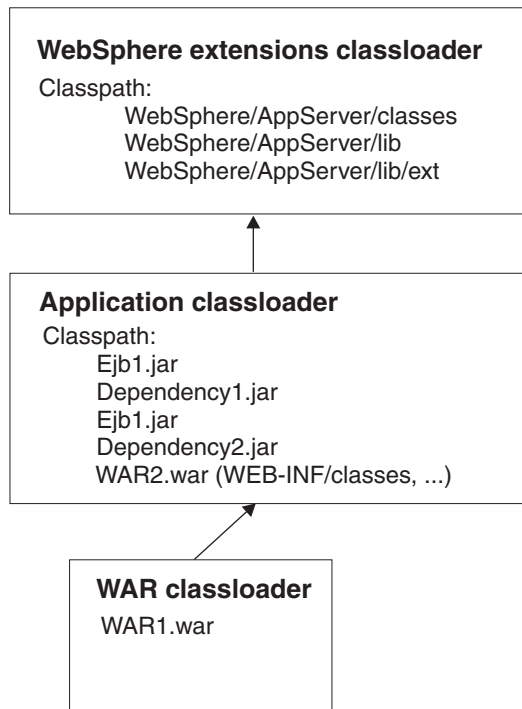
Application class-loader policy: SINGLE

```
Application 1
Module: EJB1.jar
Module: WAR1.war
  MANIFEST Class-Path: Dependency1.jar
  WAR Classloader Policy = MODULE
Application 2
```

```

Module: EJB2.jar
MANIFEST Class-Path: Dependency2.jar
Module: WAR2.war
WAR Classloader Policy = APPLICATION

```



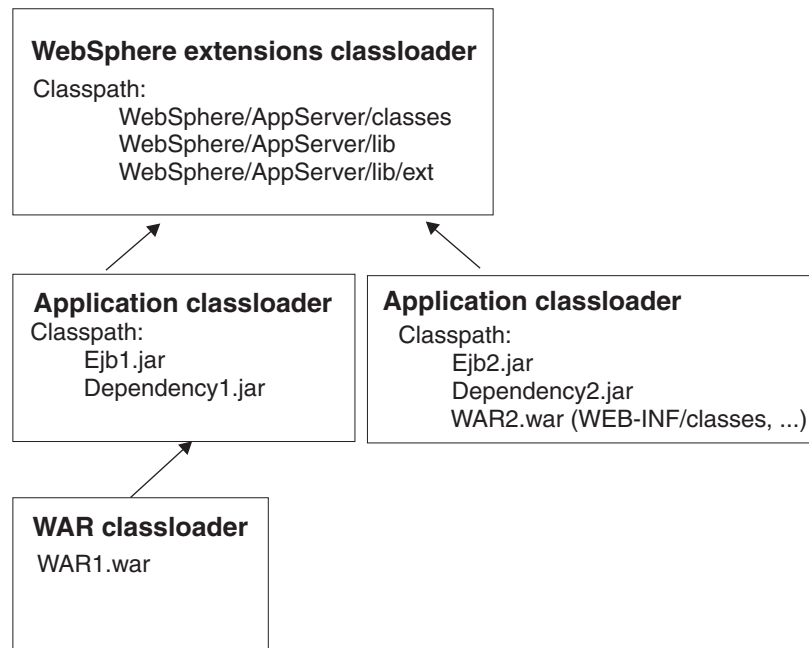
The following example shows that when the application class-loader policy of an application server is set to `MULTIPLE`, each application on the server has its own class loader. An application class loader also loads its Web modules if the application's WAR class-loader policy is set to `APPLICATION`. If the policy is set to `MODULE`, then a Web module uses its own class loader.

Application class-loader policy: `MULTIPLE`

```

Application 1
Module: EJB1.jar
Module: WAR1.war
MANIFEST Class-Path: Dependency1.jar
WAR Classloader Policy = MODULE
Application 2
Module: EJB2.jar
MANIFEST Class-Path: Dependency2.jar
Module: WAR2.war
WAR Classloader Policy = APPLICATION

```



Class-loader modes

There are two possible values for a class-loader mode:

PARENT_FIRST

The PARENT_FIRST class-loader mode causes the class loader to first delegate the loading of classes to its parent class loader before attempting to load the class from its local classpath. This is the default for class-loader policy and for standard JVM class loaders.

PARENT_LAST

The PARENT_LAST class-loader mode causes the class loader to first attempt to load classes from its local classpath before delegating the class loading to its parent. This policy allows an application class loader to override and provide its own version of a class that exists in the parent class loader.

The following settings determine a class loader's mode:

- If the application class-loader policy of an application server is SINGLE, the application class-loader policy of an application server defines the mode for an application class loader.
- If the application class-loader policy of an application server is MULTIPLE, the class-loader mode of an application defines the mode for an application class loader.
- If the WAR class-loader policy of an application is MODULE, the WAR class-loader policy of a Web module defines the mode for a WAR class loader.

Class loader collection

Use this page to manage class-loader instances on an application server. A class loader determines whether an application class loader or a parent class loader finds and loads Java class files for an application.

To view this administrative console page, click **Servers > Application Servers > *server_name* > Classloader**.

ClassLoader ID

States a string unique to the server identifying the class-loader instance. The product assigns the identifier.

ClassLoader Mode

Specifies the class-loader mode when the application class-loader policy is SINGLE. PARENT_FIRST causes the class loader to delegate the loading of classes to its parent class loader before attempting to load the class from its local classpath. PARENT_LAST causes the class loader to attempt to load classes from its local classpath before delegating the class loading to its parent; this allows an application class loader to override and provide its own version of a class that exists in the parent class loader.

Class loader settings

Use this page to configure a class loader for applications that reside on an application server.

To view this administrative console page, click **Servers > Application Servers > *server_name* > Classloader > *class_loader_ID***.

ClassLoader ID

States a string unique to the server identifying the class-loader instance. The product assigns the identifier.

Data type String

ClassLoader Mode

Specifies the class-loader mode when the application class-loader policy is SINGLE. PARENT_FIRST causes the class loader to delegate the loading of classes to its parent class loader before attempting to load the class from its local classpath. PARENT_LAST causes the class loader to attempt to load classes from its local classpath before delegating the class loading to its parent; this allows an application class loader to override and provide its own version of a class that exists in the parent class loader.

Data type String
Default PARENT_FIRST

Migrating the class-loader Module Visibility Mode setting

WebSphere Application Server Version 4.0.x had a server-wide configuration setting called **Module Visibility Mode**. For Version 5.0, you use application or WAR class-loader policies instead of module visibility modes. The Version 5.0 policies provide additional flexibility because you can configure applications running in a server for an application class-loader policy of SINGLE or MULTIPLE and for a WAR class-loader policy of APPLICATION or MODULE.

To migrate module visibility modes in your Version 4.0.x applications to their equivalents in Version 5.0, change the settings for your Version 4.0.x applications and modules to the Version 5.0 values shown in the table below.

Version 4.0.x module visibility mode	Version 5.0 application class-loader policy	Version 5.0 WAR class-loader policy
Server	SINGLE	APPLICATION

Compatibility	SINGLE	MODULE
Application	MULTIPLE	APPLICATION
Module*	MULTIPLE	MODULE
J2EE	MULTIPLE	MODULE

*There is no exact equivalent for the Version 4.0.x Module mode because it isolated EJB modules within an application.

Class loading: Resources for learning

Use the following links to find relevant supplemental information about class loaders. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

View links to additional information about:

- Programming model and decisions
- Programming instructions and examples
- Programming specifications

Programming model and decisions

- J2EE Class Loading Demystified
- Understanding J2EE Application Server Class Loading Architectures

Programming instructions and examples

- Developing and Deploying Modular J2EE Applications with WebSphere Studio Application Developer and WebSphere Application Server
- IBM WebSphere Application Server Programming

Programming specifications

- Sun's J2EE™ Platform Specification
- Sun's J2EE™ Extension Mechanism Architecture

Chapter 10. Using EJB query

The EJB query language is used to specify a query over container-managed entity beans. The language is similar to SQL. An EJB query is independent of the bean's mapping to a persistent store.

An EJB query can be used in three situations:

- To define a finder method of an EJB entity bean.
- To define a select method of an EJB entity bean.
- To dynamically specify a query using the `executeQuery()` dynamic API.

Finder and select queries are specified in the bean's deployment descriptor using the `<ejb-ql>` tag. Queries specified in the deployment descriptor are compiled into SQL during deployment. Dynamic queries require the interface provided by WebSphere Application Server Enterprise.

WebSphere's EJB query language is compliant with the EJB QL defined in Sun's EJB 2.0 specification and has additional capabilities as listed in the topic [Comparison of EJB 2.0 specification and WebSphere Query Language](#).

In your WebSphere application, you can define an EJB query in the following ways:

- **Application Assembly Tool.** When assembling an EJB 2.0 entity bean, specify the `<ejb-ql>` tag for the `finder()` or `select()` method.
- **WebSphere Studio Application Developer.** When defining an entity bean, specify the `<ejb-ql>` tag for the `finder` or `select` method.
- **Dynamic query service.** Add the `executeQuery()` method to your application. The dynamic query API is provided as an Enterprise Extension to WebSphere Application Server.

Before using EJB query, familiarize yourself with query language concepts, starting with the topic, [EJB Query Language](#).

See the topic [Example: EJB queries](#).

EJB query language

An EJB query is a string that contains the following elements:

- a `SELECT` clause that specifies the EJBs or values to return;
- a `FROM` clause that names the bean collections;
- an optional `WHERE` clause that contains search predicates over the collections;
- an optional `GROUP BY` and `HAVING` clause (see [Aggregation functions](#));
- an optional `ORDER BY` clause that specifies the ordering of the result collection.

The `SELECT` clause is optional in order to maintain compatibility with WebSphere Application Server Version 4.

Collections of entity beans are identified in EJB queries through the use of their abstract schema name in the query `FROM` clause.

The elements of EJB query language are discussed in more detail in the following related topics.

Example: EJB queries

Here is an example EJB schema, followed by a set of example queries:

Table 1. DeptBean schema

Entity bean name (EJB name)	DeptEJB (not used in query)
Abstract schema name	DeptBean
Implementation class	com.acme.hr.deptBean (not used in query)
Persistent attributes (cmp fields)	<ul style="list-style-type: none">deptno - Integer (key)name - Stringbudget - BigDecimal
Relationships	<ul style="list-style-type: none">emps - 1:Many with EmpEJBmgr - Many:1 with EmpEJB

Table 2. EmpBean schema

Entity bean name (EJB name)	EmpEJB (not used in query)
Abstract schema name	EmpBean
Implementation class	com.acme.hr.empBean (not used in query)
Persistent attributes (cmp fields)	<ul style="list-style-type: none">empid - Integer (key)name - Stringsalary - BigDecimalbonus - BigDecimalhireDate - java.sql.DatebirthDate - java.util.Calendaraddress - com.acme.hr.Address
Relationships	<ul style="list-style-type: none">dept - Many:1 with DeptEJBmanages - 1:Many with DeptEJB

Address is a serializable object used as cmp field in EmpBean. The definition of address is as follows:

```
public class com.acme.hr.Address extends Object implements Serializable {
    public String street;
    public String state;
    public String city;
    public Integer zip;
    public double distance (String start_location) { ... } ;
    public String format ( ) { ... } ;
}
```

The following query returns all departments:

```
SELECT OBJECT(d) FROM DeptBean d
```

The following query returns departments whose name begins with the letters "Web". Sort the result by name:

```
SELECT OBJECT(d) FROM DeptBean d WHERE d.name LIKE 'Web%' ORDER BY d.name
```

The keywords SELECT and FROM are shown in uppercase in the examples but are case insensitive. If a name used in a query is a reserved word, the name must be enclosed in double quotes to be used in the query. There is a list of reserved words later in this document. Identifiers enclosed in double quotes are case sensitive. This example shows how to use a cmp field that is a reserved word:

```
SELECT OBJECT(d) FROM DeptBean d WHERE d."select" > 5
```

The following query returns all employees who are managed by Bob. This example shows how to navigate relationships using a path expression:

```
SELECT OBJECT (e) FROM EmpBean e WHERE e.dept.mgr.name='Bob'
```

A query can contain a parameter which refers to the corresponding value of the finder or select method. Query parameters are numbered starting with 1:

```
SELECT OBJECT (e) FROM EmpBean e WHERE e.dept.mgr.name= ?1
```

This query shows navigation of a multivalued relationship and returns all departments that have an employee that earns at least 50000 but not more than 90000:

```
SELECT OBJECT(d) FROM DeptBean d, IN (d.emps) AS e
WHERE e.salary BETWEEN 50000 and 90000
```

There is a join operation implied in this query between each department object and its related collection of employees. If a department has no employees, the department does not appear in the result. If a department has more than one employee that earns more than 50000, that department appears multiple times in the result.

The following query eliminates the duplicate departments:

```
SELECT DISTINCT OBJECT(d) from DeptBean d, IN (d.emps) AS e WHERE e.salary > 50000
```

Find employees whose bonus is more than 40% of their salary:

```
SELECT OBJECT(e) FROM EmpBean e where e.bonus > 0.40 * e.salary
```

Find departments where the sum of salary and bonus of employees in the department exceeds the department budget:

```
SELECT OBJECT(d) FROM DeptBean d where d.budget <
( SELECT SUM(e.salary+e.bonus) FROM IN(d.emps) AS e )
```

A query can contain DB2 style date-time arithmetic expressions if you use java.sql.* datatypes as CMP fields and your datastore is DB2. Find all employees who have worked at least 20 years as of January 1st, 2000:

```
SELECT OBJECT(e) FROM EmpBean e where year( '2000-01-01' - e.hireDate ) >= 20
```

If the datastore is not DB2 or if you prefer to use java.util.Calendar as the CMP field, then you can use the java millisecond value in queries. The following query finds all employees born before Jan 1, 1990:

```
SELECT OBJECT(e) FROM EmpBean e WHERE e.birthDate < 631180800232
```

Find departments with no employees:

```
SELECT OBJECT(d) from DeptBean d where d.emps IS EMPTY
```

Find all employees whose earn more than Bob:

```
SELECT OBJECT(e) FROM EmpBean e, EmpBean b
WHERE b.name = 'Bob' AND e.salary + e.bonus > b.salary + b.bonus
```

Find the employee with the largest bonus:

```
SELECT OBJECT(e) from EmpBean e WHERE e.bonus =
(SELECT MAX(e1.bonus) from EmpBean e1)
```

The above queries all return EJB objects. A finder method query must always return an EJB Object for the home. A select method query can in addition return CMP fields or other EJB Objects not belonging to the home.

The following would be valid select method queries for EmpBean. Return the manager for each department:

```
SELECT d.mgr FROM DeptBean d
```

Return department 42 manager's name:

```
SELECT d.mgr.name FROM DeptBean d WHERE d.deptno = 42
```

Return the names of employees in department 42:

```
SELECT e.name FROM EmpBean e WHERE e.dept.deptno=42
```

Another way to write the same query is:

```
SELECT e.name from DeptBean d, IN (d.emps) AS e WHERE d.deptno=42
```

Finder and select queries allow only a single CMP field or EJBObject in the SELECT clause.

The dynamic query api allows multiple expressions in the SELECT clause. The following query would be a valid dynamic query, but not a valid select or finder query:

```
SELECT e.name, e.salary+e.bonus as total_pay , object(e), e.dept.mgr
FROM EmpBean e
ORDER BY 2
```

The following dynamic query returns the number of employees in each department:

```
SELECT e.dept.deptno as department_number , count(*) as employee_count
FROM EmpBean e
GROUP BY by e.dept.deptno
ORDER BY 1
```

The dynamic query api allows queries that contain bean or value object methods:

```
SELECT object(e), e.address.format( )
FROM EmpBean e EmpBean e
```

FROM clause

The FROM clause specifies the collections of objects to which the query is to be applied. Each collection is identified either by an abstract schema name and an identification variable, called a range variable, or by a collection member declaration that identifies a multivalued relationship and an identification variable.

Conceptually, the semantics of the query is to first form a temporary collection of tuples R. Tuples are composed of elements from the collections identified in the FROM clause. Each tuple contains one element from each of the collections in the FROM clause. All possible combinations are formed subject to the constraints imposed by the collection member declarations. If any schema name identifies a collection for which there are no records in the persistent store, then the temporary collection R will be empty.

Example: FROM clause

DeptBean contains records 10, 20 and 30 in the persistent store. EmpBean contains records 1, 2 and 3 that are related to department 10 and records 4, 5 that are related to department 20. Department 30 has no related employees.

```
FROM DeptBean d, EmpBean e
```

This forms a temporary collection R that contains 15 tuples.

```
FROM DeptBean d, DeptBean d1
```

This forms a temporary collection R that contains 9 tuples.

```
FROM DeptBean d, IN (d.emps) AS e
```

This forms a temporary collection R that contains 5 tuples. Department 30 because it contains no employees will not be in R. Department 10 will be contained in R three times and department 20 will be contained in R twice.

After forming the temporary collection the search conditions of the WHERE clause will be applied to R and this will yield a new temporary collection R1. The ORDER BY and SELECT clauses are applied to R1 to yield the final result set.

An identification variable is a variable declared in the FROM clause using the operator IN or the optional AS.

```
FROM DeptBean AS d, IN (d.emps) AS e
```

is equivalent to:

```
FROM DeptBean d, IN (d.emps) e
```

An identification variable that is declared to be an abstract schema name is called a range variable. In the query above "d" is a range variable. An identification variable that is declared to be a multivalued path expression is called a collection member declaration. "d" and "e" in the example above are collection member declarations.

Note that the following path expression is illegal as a collection member declaration because it is not multivalued:

```
e.dept.mgr
```

Inheritance in EJB query

If an EJB inheritance hierarchy has been defined for an abstract schema, using the abstract schema name in a query statement implies the collection of objects for that abstract schema as well as all subtypes.

Example: Inheritance

Suppose that bean ManagerBean is defined as a subtype of EmpBean and ExecutiveBean is a subtype of ManagerBean in an EJB inheritance hierarchy. The following query returns employees as well as managers and executives:

```
SELECT OBJECT(e) FROM EmpBean e
```

Path expressions

An identification variable followed by the navigation operator (.) and a cmp or relationship name is a path expression.

A path expression that leads to a cmr field can be further navigated if the cmr field is single-valued. If the path expression leads to a multi-valued relationship, then the path expression is terminal and cannot be further navigated. If the path expression leads to a cmp field whose type is a value object, it is possible to navigate to attributes of the value object.

Example: Value object

Assume that address is a cmp field for EmpBean, which is a value object.

```
SELECT object(e) FROM EmpBean e
WHERE e.address.distance('San Jose') < 10 and e.address.zip = 95037
```

It is best to use the composer pattern to map value object attributes to relational columns if you intend to search on value attributes. If you store value objects in serialized format, then each value object must be retrieved from the database and deserialized. Value object methods can only be done in dynamic queries.

A path expression can also navigate to a bean method. The method must be defined on either the remote or local bean interface. Methods can only be used in dynamic queries. You cannot mix both remote and local methods in a single query statement.

If the query contains remote methods, the dynamic query must be executed using the query remote interface. Using the query remote interface causes the query service to activate beans and create instances of the remote bean interface

Likewise, a query statement with local bean methods must be executed with the query local interface. This causes the query service to activate beans and local interface instances.

Do not use get methods to access cmp and cmr fields of a bean.

If a method has overloaded definitions, the overloaded methods must have different number of parameters.

Methods must have non-void return types and method arguments and return types must be either primitive types byte, short, int, long, float, double, boolean, char or wrapper types from the following list:

Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Boolean, Character, java.util.Calendar, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.util.Date

If any input argument to a method is NULL, it is assumed the method returns a NULL value and the method is not invoked.

A collection valued path expression can be used in the FROM clause as a collection member declaration, and with the IS EMPTY, MEMBER OF, and EXISTS predicates in the WHERE clause.

FROM EmpBean e WHERE e.dept.mgr.name='Bob'	OK
FROM EmpBean e WHERE e.dept.emps.name='BOB'	INVALID -- cannot navigate through emps because it is multivalued
FROM EmpBean e, IN (e.dept.emps) e1 WHERE e1.name='BOB'	OK
FROM EmpBean e WHERE e.dept.emps IS EMPTY	OK

WHERE clause

The WHERE clause contains search conditions composed of the following:

- literal values
- input parameters
- expressions

- basic predicates
- quantified predicates
- BETWEEN predicate
- IN predicate
- LIKE predicate
- NULL predicate
- EMPTY collection predicate
- MEMBER OF predicate
- EXISTS predicate
- IS OF TYPE predicate

If the search condition evaluates to TRUE, the tuple is added to the result set.

Literals

A string literal is enclosed in single quotes. A single quote that occurs within a string literal is represented by two single quotes; For example: 'Tom''s'. A string literal cannot exceed the maximum length that is supported by the underlying persistent data store.

A numeric literal can be any of the following:

- an exact value such as 57, -957, +66
- any value supported by Java long
- a decimal literal such as 57.5, -47.02
- an approximate numeric value such as 7E3, -57.4E-2

A decimal or approximate numeric value must be in the range supported by the underlying persistent data store.

A boolean literal can be the keyword TRUE or FALSE and is case insensitive.

Input parameters

Input parameters are designated by the question mark followed by a number; For example: ?2

Input parameters are numbered starting at 1 and correspond to the arguments of the finder or select method; therefore, a query must not contain an input parameter that exceeds the number of input arguments.

An input parameter can be a primitive type of byte, short, int, long, float, double, boolean, char or wrapper types of Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Boolean, Char, java.util.Calendar, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp or an EJBObject.

An input parameter must not have a NULL value. To search for the occurrence of a NULL value the NULL predicate should be used.

Expressions

Conditional expressions can consist of comparison operators and logical operators (AND, OR, NOT).

Arithmetic expressions can be used in comparison expressions and can be composed of arithmetic operations and functions, path expressions that evaluate to a numeric value and numeric literals and numeric input parameters.

String expressions can be used in comparison expressions and can be composed of string functions, path expressions that evaluate to a string value and string literals and string input parameters. A cmp field of type char is handled as if it were a string of length 1.

Boolean expressions can be used with = and <> comparison and can be composed of path expressions that evaluate to a boolean value and TRUE and FALSE keywords and boolean input parameters.

Reference expressions can be used with = and <> comparison and can be composed of path expressions that evaluate to a cmr field, an identification variable and an input parameter whose type is an EJB reference

Four different expression types are supported for working with date-time types. For portability the java.util.Calendar type should be used. DB2 style date, time and timestamp expressions are supported if the datastore is DB2 and the CMP field is of type java.util.Date, java.sql.Date, java.sql.Time or java.sql.Timestamp.

A Calendar type can be compared to another Calendar type, an exact numeric literal or input parameter of type long whose value is the standard Java long millisecond value.

The following query finds all employees born before Jan 1, 1990:
SELECT OBJECT(e) FROM EmpBean e WHERE e.birthDate < 631180800232

Date expressions can be used in comparison expressions and can be composed of operators + - , date duration expressions and date functions, path expressions that evaluate to a date value, string representation of a date and date input parameters.

Time expressions can be used in comparison expressions and can be composed of operators + - , time duration expressions and time functions, path expressions that evaluate to a time value, string representation of time and time input parameters.

Timestamp expressions can be used in comparison expressions and can be composed of operators + - , timestamp duration expressions and timestamp functions, path expressions that evaluate to a timestamp value, string representation of a timestamp and timestamp input parameters.

Standard bracketing () for ordering expression evaluation is supported.

The operators and their precedence order from highest to lowest are:

- Navigation operator (.)
- Arithmetic operators in precedence order:
 - + - unary
 - * / multiply, divide
 - + - add, subtract
- Comparison operators: =, >, <, >=, <=, <>(not equal)
- Logical operator NOT
- Logical operator AND
- Logical operator OR

In some datastores, a zero length string value ('') is treated as a null value and affects the results of queries. Some datastores perform division between two integer values using integer arithmetic rules and other datastores use non integer rules. This also can affect the results of queries. For portability, avoid the use of zero length string values and division of integer values in an EJB query.

Null value semantics: The following describe the semantics of NULL values:

- Comparison or arithmetic operations with an unknown (NULL) value yield an unknown value
- Path expressions that contain NULL evaluate to NULL
- The IS NULL and IS NOT NULL operators can be applied to path expressions and return TRUE or FALSE. Boolean operators AND, OR and NOT use three valued logic.

AND	True	False	Unknown
True	True	False	Unknown
False	False	False	False
Unknown	Unknown	False	Unknown

OR	True	False	Unknown
True	True	True	True
False	True	False	Unknown
Unknown	True	Unknown	Unknown

	NOT
True	False
False	True
Unknown	Unknown

Example: Null value semantics

```
select object(e) from EmpBean where e.salary > 10 and e.dept.budget > 100
```

If salary is NULL the evaluation of `e.salary > 10` returns unknown and the employee object is not returned. If the cmr field `dept` or `budget` is NULL evaluation of `e.dept.budget > 100` returns unknown and the employee object is not returned.

```
select object(e) from EmpBean where e.dept.budget is null
```

If `dept` or `budget` is NULL evaluation of `e.dept.budget is null` returns TRUE and the employee object is returned.

```
select object(e) from EmpBean e , in (e.dept.emps) e1 where e1.salary > 10
```

If `dept` is NULL, then the multivalued path expression `e.dept.emps` results in an empty collection (not a collection that contains a NULL value). An employee with a null `dept` value will not be returned.

```
select object(e) from EmpBean e where e.dept.emps is empty
```

If `dept` is NULL the evaluation of the predicate in unknown and the employee object is not returned.

```
select object(e) from EmpBean e , EmpBean e1 where e member of e1.dept.emps
```

If `dept` is NULL evaluation of the `member of` predicate returns unknown and the employee is not returned.

Date time arithmetic and comparisons: DATE, TIME and TIMESTAMP values may be compared with another value of the same type. Comparisons are chronological. Date time values can also be incremented, decremented, and subtracted.

If the datastore is DB2, then DB2 string representation of DATE, TIME and TIMESTAMP types can also be used. A string representation of a date or time can use ISO, USA, EUR or JIS format. A string representation of a timestamp uses ISO format.

Format	Date format	Date examples	Time format	Time examples
ISO	yyyy-mm-dd	1987-02-24 1987-2-24	hh.mm.ss	13.50.00 13.50
USA	mm/dd/yyyy	2/24/1987	hh:mm AM or PM	1:50 pm 02:10 AM
EUR	dd.mm.yyyy	24.02.1987 24.2.1987	hh.mm.ss	13.50.00 13.55
JIS	yyyy-mm-dd	1987-02-24	hh:mm:ss	13:50 13:50:05

Example 1: Date time arithmetic comparisons

```
e.hiredate > '1990-02-24'
```

The timestamp of February 24th, 1990 1:50 pm can be represented as follows:

```
'1990-02-24-13.50.00.000000' or  
'1990-02-24-13.50.00'
```

If the datastore is DB2, DB2 decimal durations can be used in expressions and comparisons. A date duration is a decimal(8,0) number that represents the difference between two dates in the format YYYYMMDD. A time duration is a decimal(6,0) number that represents the difference between two time values as HHMMSS. A timestamp duration is a decimal(20,6) number representing the differences between two timestamp values as YYYYMMDDHHMMSS.ZZZZZZ (ZZZZZZ is the number of microseconds and is to the right of the decimal point).

Two date values (or time values or timestamp values) can be subtracted to yield a duration. If the second operand is greater than the first the duration is a negative decimal number. A duration can be added or subtracted from a datetime value to yield a new datetime value.

Example 2: Date time arithmetic comparisons

DATE('3/15/2000') - '12/31/1999' results in a decimal number 215 which is a duration of 0 years, 2 months and 15 days.

Durations are really decimal numbers and can be used in arithmetic expressions and comparisons.

```
( DATE('3/15/2000') - '12/31/1999' ) + 14 > 215 evaluates to TRUE.
```

DATE('12/31/1999') + DECIMAL(215,8,0) results in a date value 3/15/2000.

TIME('11:02:26') - '00:32:56' results in a decimal number 102930 which is a time duration of 10 hours, 29 minutes and 30 seconds.

`TIME('00:32:56') + DECIMAL(102930,6,0)` results in a time value of 11:02:26.

`TIME('00:00:59') + DECIMAL(240000,6,0)` results in a time value of 00:00:59.

`e.hiredate + DECIMAL(500,8,0) > '2000-10-01'` means compare the hiredate plus 5 months to the date 10/01/2000.

Basic predicates

Basic predicates can be of two forms

```
expression-1 comparison-operator expression-2
expression-3 comparison-operator ( subselect )
```

The subselect must not return more than one value and the subselect can not return a type of an EJB reference. Boolean types and reference types only support = and <> comparisons.

Example: Basic predicates

```
d.name='Java Development'
e.salary > 20000
e.salary > ( select avg(e.salary) from EmpBean e)
```

Quantified predicates

A quantified predicate compares a value with a set of values produced by a subselect.

```
expression comparison-operator SOME | ANY | ALL ( subselect )
```

The expression must not evaluate to a reference type.

When SOME or ANY is specified the result of the predicate is as follows:

- TRUE if the comparison is true for at least one value returned by the subselect.
- FALSE if the subselect is empty or if the comparison is false for every value returned by the subselect.
- UNKNOWN if the comparison is not true for all of the values returned by the subselect and at least one of the comparisons is unknown because of a null value.

When ALL is specified the result of the predicate is as follows:

- TRUE if the subselect returns empty or if the comparison is true to every value returned by the subselect.
- FALSE if the comparison is false for at least one value returned by the subselect.
- UNKNOWN if the comparison is not false for all values returned by the subselect and at least one comparison is unknown because of a null value.

BETWEEN predicate

The BETWEEN predicate determines whether a given value lies between two other given values.

```
expression [NOT] BETWEEN expression-2 AND expression-3
```

Example: BETWEEN predicate

```
e.salary BETWEEN 50000 AND 60000
```

is equivalent to:

```
e.salary >= 50000 AND e.salary <= 60000
e.name NOT BETWEEN 'A' AND 'B'
```

is equivalent to:

```
e.name < 'A' OR e.name > 'B'
```

IN predicate

The IN predicate compares a value to a set of values and can have one of two forms:

```
expression [NOT] IN ( subselect )
expression [NOT] IN ( value1, value2, .... )
```

ValueN can either be a literal value or an input parameter. The expression can not evaluate to a reference type.

Example: IN predicate

```
e.salary IN ( 10000, 15000 )
```

is equivalent to

```
( e.salary = 10000 OR e.salary = 15000 )
e.salary IN ( select e1.salary from EmpBean e1 where e1.dept.deptno = 10)
```

is equivalent to

```
e.salary = ANY ( select e1.salary from EmpBean e1 where e1.dept.deptno = 10)
e.salary NOT IN ( select e1.salary from EmpBean e1 where e1.dept.deptno = 10)
```

is equivalent to

```
e.salary <> ALL ( select e1.salary from EmpBean e1 where e1.dept.deptno = 10)
```

LIKE predicate

The LIKE predicate searches a string value for a certain pattern.

```
string-expression [NOT] LIKE pattern [ ESCAPE escape-character ]
```

The pattern value is a string literal or parameter marker of type string in which the underscore (`_`) stands for any single character and percent (`%`) stands for any sequence of characters (including empty sequence). Any other character stands for itself. The escape character can be used to search for character `_` and `%`. The escape character can be specified as a string literal or an input parameter.

If the string-expression is null, then the result is unknown.

If both string-expression and pattern are empty, then the result is true.

Example: LIKE predicate

- `'' LIKE ''` is true
- `'' LIKE '%'` is true
- `e.name LIKE '12%3'` is true for `'123'` `'12993'` and false for `'1234'`
- `e.name LIKE 's_me'` is true for `'some'` and `'same'`, false for `'soome'`
- `e.name LIKE '/_foo'` escape `'/'` is true for `'_foo'`, false for `'afoo'`
- `e.name LIKE '//_foo'` escape `'/'` is true for `'/afoo'` and for `'/bfoo'`
- `e.name LIKE '///_foo'` escape `'/'` is true for `'/_foo'` but false for `'/afoo'`

NULL predicate

The NULL predicate tests for null values.

```
single-valued-path-expression IS [NOT] NULL
```

Example: NULL predicate

```
e.name IS NULL
e.dept.name IS NOT NULL
```

e.dept IS NOT NULL

EMPTY collection predicate

To test if a multivalued relationship is empty, use the following syntax:

collection-valued-path-expression IS [NOT] EMPTY

Example: Empty collection predicate

To find all departments with no employees:

```
SELECT OBJECT(d) FROM DeptBean d WHERE d.emps IS EMPTY
```

MEMBER OF predicate

This expression tests whether the object reference specified by the single valued path expression or input parameter is a member of the designated collection. If the collection valued path expression designates an empty collection the value of the MEMBER OF expression is FALSE.

{ single-valued-path-expression | input_parameter } [NOT] MEMBER [OF] collection-valued-path-expression

Example: MEMBER OF predicate

Find employees that are not members of a given department number:

```
SELECT OBJECT(e) FROM EmpBean e , DeptBean d
WHERE e NOT MEMBER OF d.emps AND d.deptno = ?1
```

Find employees whose manager is a member of a given department number:

```
SELECT OBJECT(e) FROM EmpBean e, DeptBean d
WHERE e.dept.mgr MEMBER OF d.emps and d.deptno=?1
```

EXISTS predicate

The exists predicate tests for the presence or absence of a condition specified by a subselect.

EXISTS (subselect)

EXISTS collection-valued-path-expression

The result of EXISTS is true if the subselect returns at least one value or the path expression evaluates to a nonempty collection, otherwise the result is false.

To negate an EXISTS predicate, precede it with the logical operator NOT.

Example: EXISTS predicate

Return departments that have at least one employee earning more than 1000000:

```
SELECT OBJECT(d) FROM DeptBean d
WHERE EXISTS ( SELECT 1 FROM IN (d.emps) e WHERE e.salary > 1000000 )
```

Return departments that have no employees:

```
SELECT OBJECT(d) FROM DeptBean d
WHERE NOT EXISTS ( SELECT 1 FROM IN (d.emps) e)
```

The above query can also be written as follows:

```
SELECT OBJECT(d) FROM DeptBean d WHERE NOT EXISTS d.emps
```

IS OF TYPE predicate

The IS OF TYPE predicate is used to test the type of an EJB reference. It is similar in function to the Java instance of operator. IS OF TYPE is used when several abstract beans have been grouped into an EJB inheritance hierarchy. The type

names specified in the predicate are the bean abstract names. The ONLY option can be used to specify that the reference must be exactly this type and not a subtype.

```
identification-variable IS OF TYPE ( [ONLY] type-1, [ONLY] type-2, ..... )
```

Example: IS OF TYPE predicate

Suppose that bean ManagerBean is defined as a subtype of EmpBean and ExecutiveBean is a subtype of ManagerBean in an EJB inheritance hierarchy.

The following query returns employees as well as managers and executives:

```
SELECT OBJECT(e) FROM EmpBean e
```

If you are interested in objects which are employees and not managers and not executives:

```
SELECT OBJECT(e) FROM EmpBean e WHERE e IS OF TYPE( ONLY EmpBean )
```

If you are interested in object which are managers or executives:

```
SELECT OBJECT(e) FROM EmpBean e WHERE e IS OF TYPE( ManagerBean)
```

The above query is equivalent to the following query:

```
SELECT OBJECT(e) FROM ManagerBean e
```

If you are interested in managers only and not executives:

```
SELECT OBJECT(e) FROM EmpBean e WHERE e IS OF TYPE( ONLY ManagerBean)
```

or:

```
SELECT OBJECT(e) FROM ManagerBean e  
WHERE e IS OF TYPE (ONLY ManagerBean)
```

Scalar functions

EJB query contains scalar built-in functions for doing type conversions, string manipulation, and for manipulating date-time values. The list of scalar functions is documented in the topic EJB query: Scalar functions.

Example: Scalar functions

Find employees hired in 1999:

```
SELECT OBJECT(e) FROM EmpBean e where YEAR(e.hireDate) = 1999
```

The only scalar functions that are guaranteed to be portable across backend datastore vendors are the following:

- ABS
- SQRT
- CONCAT
- LENGTH
- LOCATE
- SUBSTRING

The other scalar functions should be used only when DB2 is the backend datastore.

EJB query: Scalar functions

EJB query contains scalar built-in functions, as listed below, for doing type conversions, string manipulation, and for manipulating date-time values.

Numeric functions

ABS (< any numeric datatype >) -> < any numeric datatype >

SQRT (< any numeric datatype >) -> Double

Type conversion functions

CHAR (< any numeric datatype >) -> string

CHAR (< string >) -> string

CHAR (< any datetime datatype > [, Keyword k]) -> string

Datetime datatype is converted to its string representation in a format specified by the keyword k. The valid keywords values are ISO, USA, EUR or JIS. If k is not specified the default is ISO.

BIGINT (< any numeric datatype >) -> Long

BIGINT (< string >) -> Long

The following function converts the argument to an integer n by truncation and returns the date that is n-1 days after January 1, 0001:

DATE (< date string >) -> Date

DATE (< any numeric datatype>) -> Date

The following function returns date portion of a timestamp:

DATE(timestamp) -> Date

DATE (< timestamp-string >) -> Date

The following function converts number to decimal with optional precision p and scale s.

DECIMAL (< any numeric datatype > [, p [,s]]) -> Decimal

The following function converts string to decimal with optional precision p and scale s.

DECIMAL (< string > [, p [, s]]) -> Decimal

DOUBLE (< any numeric datatype >) -> Double

DOUBLE (< string >) -> Double

FLOAT (< any numeric datatype >) -> Double

FLOAT (< string >) -> Double

Float is a synonym for DOUBLE.

INTEGER (< any numeric datatype >) -> Integer

INTEGER (< string >) -> Integer

REAL (< any numeric datatype >) -> Float

SMALLINT (< any numeric datatype >) -> Short

SMALLINT (< string >) -> Short

TIME (< time >) -> Time

TIME (< time-string >) -> Time

TIME (< timestamp >) -> Time

TIME (< timestamp-string >) -> Time

TIMESTAMP (< timestamp >) -> Timestamp

TIMESTAMP (< timestamp-string >) -> Timestamp

String functions

CONCAT (<string>, <string>) -> String

The following function returns a character string representing absolute value of the argument not including its sign or decimal point. For example, digits(-42.35) is "4235".

DIGITS (Decimal d) -> String

The following function returns the length of the argument in bytes. If the argument is a numeric or datetime type, it returns the length of internal representation.

LENGTH (< string >) -> Integer

The following function returns a copy of the argument string where all upper case characters have been converted to lower case.

LCASE (< string >) -> String

The following function returns the starting position of the first occurrence of argument 1 inside argument 2 with optional start position. If not found, it returns 0.

LOCATE (String s1 , String s2 [, Integer start]) -> Integer

The following function returns a substring of s beginning at character m and containing n characters. If n is omitted, the substring contains the remainder of string s. The result string is padded with blanks if needed to make a string of length n.

SUBSTRING (String s , Integer m [, Integer n]) -> String

The following function returns a copy of the argument string where all lower case characters have been converted to upper case.

UCASE (< string >) -> String

Date - time functions

The following function returns the day portion of its argument. For a duration, the return value can be -99 to 99.

DAY (Date) -> Integer
DAY (< date-string >) -> Integer
DAY (< date-duration >) -> Integer
DAY (Timestamp) -> Integer
DAY (< timestamp-string >) -> Integer
DAY (< timestamp-duration >) -> Integer

The following function returns one more than number of days from January 1, 0001 to its argument.

DAYS (Date) -> Integer
DAYS (< Date-string >) -> Integer
DAYS (Timestamp) -> Integer
DAYS (< timestamp-string >) -> Integer

The following function returns the hour part of its argument. For a duration, the return value can be -99 to 99.

HOUR (Time) -> Integer
HOUR (< time-string >) -> Integer
HOUR (< time-duration >) -> Integer
HOUR (Timestamp) -> Integer
HOUR (< timestamp-string >) -> Integer
HOUR (< timestamp-duration >) -> Integer

The following function returns the microsecond part of its argument.

MICROSECOND (Timestamp) -> Integer
MICROSECOND (< timestamp-string >) -> Integer
MICROSECOND (< timestamp-duration >) -> Integer

The following function returns the minute part of its argument. For a duration, the return value can be -99 to 99.

```
MINUTE ( Time ) -> Integer
MINUTE ( < time-string > ) -> Integer
MINUTE ( < time-duration > ) -> Integer
MINUTE ( Timestamp ) -> Integer
MINUTE ( < timestamp-string > ) -> Integer
MINUTE ( < timestamp-duration > ) -> Integer
```

The following function returns the month portion of its argument. For a duration, the return value can be -99 to 99.

```
MONTH ( Date ) -> Integer
MONTH ( < date-string > ) -> Integer
MONTH ( < date-duration > ) -> Integer
MONTH ( Timestamp ) -> Integer
MONTH ( < timestamp-string > ) -> Integer
MONTH ( < timestamp-duration > ) -> Integer
```

The following function returns the second part of its argument. For a duration, the return value can be -99 to 99.

```
SECOND ( Time ) -> Integer
SECOND ( < time-string > ) -> Integer
SECOND ( < time-duration > ) -> Integer
SECOND ( Timestamp ) -> Integer
SECOND ( < timestamp-string > ) -> Integer
SECOND ( < timestamp-duration > ) -> Integer
```

The following function returns the year portion of its argument. For a duration, the return value can be -9999 to 9999.

```
YEAR ( Date ) -> Integer
YEAR ( < date-string > ) -> Integer
YEAR ( < date-duration > ) -> Integer
YEAR ( Timestamp ) -> Integer
YEAR ( < timestamp-string > ) -> Integer
YEAR ( < timestamp-duration > ) -> Integer
```

Aggregation functions

Queries that return aggregate values can only be used with the dynamic query interface available in WebSphere Application Server Enterprise. However, aggregation functions can be used in non-dynamic queries if the aggregation function is used in a subselect or HAVING clause.

Aggregation functions operate on a set of values to return a single scalar value. The following is an example of an aggregation:

```
SELECT SUM (e.salary + e.bonus) FROM EmpBean e WHERE e.dept.deptno =20
```

This computes the total salary and bonus for department 20.

The aggregation functions are avg, count, max, min and sum. The syntax of an aggregation function is as follows:

```
aggregation-function ( [ ALL | DISTINCT ] expression )
```

or:

```
COUNT( * )
```

The DISTINCT option eliminates duplicate values before applying the function. ALL is the default and does not eliminate duplicates. Null values are ignored in computing the aggregate function except for COUNT(*) which returns a count of all elements in the set.

MAX and MIN can apply to any numeric, string or datetime datatype and returns the same datatype. SUM and AVG take a numeric type as input. SUM and AVG return numeric type. The actual numeric type returned by SUM and AVG depends on the underlying datastore. COUNT can take any datatype and returns an integer.

If you are using an Informix datastore, the argument to COUNT must be an asterisk or a single valued path expression. The argument to SUM, AVG, MIN, or MAX used with DISTINCT must be a single valued path expression.

The set of values that is used for the aggregate function is determined by the collection that results from the FROM and WHERE clause of the subquery. This set can be divided into groups and the aggregation function applied to each group. This is done by using a GROUP BY clause in the subquery. The GROUP BY clause defines grouping members which is a list of path expressions. Each path expression specifies a field that is a primitive type of byte, short, int, long, float, double, boolean, char, or a wrapper type of Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Boolean, Character, java.util.Calendar, java.util.Date, java.sql.Date, java.sql.Time or java.sql.Timestamp.

Finder or select queries can not return aggregation function values. In other words, aggregation functions can not appear in the top level SELECT of a finder or select query but can be used in subqueries.

Example: Aggregation functions

```
SELECT e.dept.deptno, AVG ( e.salary) FROM EmpBean e GROUP BY e.dept.deptno
```

The above query computes the average salary for each department.

In dividing a set into groups, a NULL value is considered equal to another NULL value.

Just as the WHERE clause filters tuples from the FROM clause, the groups can be filtered using a HAVING clause that tests group properties involving aggregate functions or grouping members:

```
SELECT e.dept.deptno, AVG ( e.salary) FROM EmpBean e
GROUP BY e.dept.deptno
HAVING COUNT(*) > 3 AND e.dept.deptno > 5
```

This query returns average salary for departments that have more than 3 employees and the department number is greater than 5.

It is possible to have a HAVING clause without a GROUP BY clause in which case the entire set is treated as a single group to which the HAVING clause is applied.

SELECT clause

For finder and select queries, the syntax of the SELECT clause is as follows:

```
SELECT [ ALL | DISTINCT ]
{ single-valued-path-expression | OBJECT ( identification-variable ) }
```

The SELECT clause consists of either a single identification variable that is defined in the FROM clause or a single valued path expression that evaluates to a object reference or CMP value. The keyword DISTINCT can be used to eliminate duplicate references.

For a query that defines a finder method the query must return an object type consistent with the home for which the finder method associated with the query. In other words, a finder method for a department home can not return employee objects.

For dynamic queries the syntax is as follows:

```
SELECT { ALL | DISTINCT } [ selection , ]* selection
selection ::= { expression [[AS] id ] | scalar-subselect }
```

A scalar-subselect is a subselect that returns a single value.

Example: SELECT clause

Find all employees that earn more than John:

```
SELECT OBJECT(e) FROM EmpBean ej, EmpBean e
WHERE ej.name = 'John' and e.salary > ej.salary
```

Find all departments that have one or more employees who earn less than 20000:

```
SELECT DISTINCT e.dept FROM EmpBean e where e.salary < 20000
```

A select method query can have a path expression that evaluates to an arbitrary value:

```
SELECT e.dept.name FROM EmpBean e where e.salary < 2000
```

The above query returns a collection of name values for those departments having employees earning less than 20000.

Example: Valid dynamic queries

The following are examples of dynamic queries:

```
SELECT e.name, e.salary+e.bonus as total_pay from EmpBean e
SELECT SUM( e.salary+e.bonus) from EmpBean e where e.dept.deptno = ?1
```

ORDER BY clause

The ORDER BY clause specifies an ordering of the objects in the result collection:

```
ORDER BY [ order_element ,]* order_element
order_element ::= { path-expression | integer } [ ASC | DESC ]
```

The path expression must specify a single valued field that is a primitive type of byte, short, int, long, float, double, char or a wrapper type of Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Character, java.util.Calendar, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp.

ASC specifies ascending order and is the default. DESC specifies descending order.

Integer refers to a selection expression in the SELECT clause.

Example: ORDER BY clause

Return department objects in decreasing deptno order:

```
SELECT OBJECT(d) FROM DeptBean d ORDER BY d.deptno DESC
```

Return employee objects sorted by department number and name:

```
SELECT OBJECT(e) FROM EmpBean e ORDER BY e.dept.deptno ASC, e.name DESC
```

The following is a valid dynamic query:

```
SELECT OBJECT(e), e.salary+e.bonus as total_pay FROM EmpBean e ORDER BY 2 DESC
```

Subqueries

A subquery can be used in quantified predicates, EXISTS predicate or IN predicate. A subquery should only specify a single element in the SELECT clause. When a path expression appears in a subquery, the identification variable of the path expression must be defined either in the subquery, in one of the containing subqueries, or in the outer query. A scalar subquery is a subquery that returns one value. A scalar subquery can be used in a basic predicate and in the SELECT clause of a dynamic query.

Example: Subqueries

```
SELECT OBJECT(e) FROM EmpBean e
WHERE e.salary > ( SELECT AVG(e1.salary) FROM EmpBean e1)
```

The above query returns employees who earn more than average salary of all employees.

```
SELECT OBJECT(e) FROM EmpBean e WHERE e.salary >
( SELECT AVG(e1.salary) FROM IN (e.dept.emps) e1 )
```

The above query returns employees who earn more than average salary of their department.

```
SELECT OBJECT(e) FROM EmpBean e WHERE e.salary =
( SELECT MAX(e1.salary) FROM IN (e.dept.emps) e1 )
```

The above query returns employees who earn the most in their department.

```
SELECT OBJECT(e) FROM EmpBean e
WHERE e.salary > ( SELECT AVG(e.salary) FROM EmpBean e1
WHERE YEAR(e1.hireDate) = YEAR(e.hireDate) )
```

The above query returns employees who earn more than the average of employees hired in same year.

EJB query restrictions

An EJB query is compiled into an SQL query and executed against the underlying datastore based on schema mapping of the abstract bean to the datastore schema. The semantics of comparison and arithmetic operations are that of the underlying datastore. In the case of SQL, note that two strings are equal if the shorter string padded with blanks equals the longer string. For example, 'A' is equal to 'A '. This differs from the equality of strings in the Java language. Arithmetic overflow operations are an error in SQL.

A cmp field can not be used in comparison operations or predicates (except for LIKE) if that cmp field is mapped to a long varchar or large objects (LOB) column or any other column type for which the database server does not support predicates or comparison operations.

A cmp field of any type can be used in a SELECT clause. Fields that can be used in predicates, grouping, or ordering operations must be of the types listed below:

- Primitive types : byte, short, int, long, float, double, boolean, char
- Object types: Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Boolean, Char, java.util.Calendar , java.util.Date
- JDBC types: java.sql.Date, java.sql.Time, java.sql.Timestamp

The field must be mapped to a table column that is compatible in type either by using a "top-down" default mapping generated by the WebSphere deploy tool, or using a "meet-in-the-middle" mapping between compatible types.

In order to search on attributes of a cmp field that is a user-defined value object, you should use a "meet-in-the-middle" mapping and use a composer to map each attribute to a compatible column. The default "top-down" mapping stores the object as a serialized object in a column of type blob, which does not allow searching.

If a cmp field is mapped to a column using a "meet-in-the-middle" mapping with a converter, that field can only be used with the NULL predicate or with basic predicates of the following form:

```
path-expression <comparison> literal_value
path-expression <comparison> input_parameter
```

In this situation, the converter method `toData()` is called to convert the right-hand side of the predicate to an SQL value.

Example of allowable predicate on a cmp field with user defined converter:

```
e.name = 'Chris'
e.name > ?1
e.name IS NULL
```

Examples of unallowable predicates:

```
substring( e.name, 1, 3 ) = 'ABC'
e.salary > d.budget
```

A converter should preserve equality, collating sequence and null values when doing a conversion. Otherwise cmp fields created by the converter should not be used in WHERE, GROUP, HAVING or ORDER clauses of a query.

EJB Query: Reserved words

The following words are reserved in WebSphere EJB query:

all, as, distinct, empty, false, from, group, having, in, is, like, select, true, union, where

Avoid using identifiers that start with underscore (for example, `_integer`) as these are also reserved.

EJB query: BNF syntax

```
EJB QL ::= [select_clause] from_clause [where_clause]
          [group_by_clause] [having_clause] [order_by_clause]
from_clause ::= FROM identification_variable_declaration
              [,identification_variable_declaration]*
identification_variable_declaration ::= collection_member_declaration |
                                     range_variable_declaration
```

```

collection_member_declaration ::=
    IN ( collection_valued_path_expression ) [AS] identifier
range_variable_declaration ::= abstract_schema_name [AS] identifier
single_valued_path_expression ::=
    {single_valued_navigation | identification_variable}. ( cmp_field |
        method | cmp_field.value_object_attribute | cmp_field.value_object_method )
    | single_valued_navigation
single_valued_navigation ::=
    identification_variable.[ single_valued_cmr_field. ]*
    single_valued_cmr_field
collection_valued_path_expression ::=
    identification_variable.[ single_valued_cmr_field. ]*
    collection_valued_cmr_field
select_clause ::= SELECT { ALL | DISTINCT } {single_valued_path_expression |
    identification_variable | OBJECT ( identification_variable) }
select_clause_eex ::= SELECT { ALL | DISTINCT } [ selection , ]* selection
selection ::= { expression [[AS] id ] | subselect }
order_by_clause ::= ORDER BY [ {single_valued_path_expression | integer} [ASC|DESC], ]*
    {single_valued_path_expression | integer} [ASC|DESC]
where_clause ::= WHERE conditional_expression
conditional_expression ::= conditional_term |
    conditional_expression OR conditional_term
conditional_term ::= conditional_factor |
    conditional_term AND conditional_factor
conditional_factor ::= [NOT] conditional_primary
conditional_primary ::= simple_cond_expression | (conditional_expression)
simple_cond_expression ::= comparison_expression | between_expression |
    like_expression | in_expression | null_comparison_expression |
    empty_collection_comparison_expression | quantified_expression |
    exists_expression | is_of_type_expression | collection_member_expression
between_expression ::= expression [NOT] BETWEEN expression AND expression
in_expression ::= single_valued_path_expression [NOT] IN
    { (subselect) | ( atom , ]* atom ) }
atom = { string-literal | numeric-constant | input-parameter }
like_expression ::= expression [NOT] LIKE
    {string_literal | input_parameter}
    [ESCAPE {string_literal | input_parameter}]
null_comparison_expression ::=
    single_valued_path_expression IS [ NOT ] NULL
empty_collection_comparison_expression ::=
    collection_valued_path_expression IS [NOT] EMPTY
collection_member_expression ::=
    { single_valued_path_expression | input_paramter } [ NOT ] MEMBER [ OF ]
    collection_valued_path_expression
quantified_expression ::=
    expression comparison_operator {SOME | ANY | ALL} (subselect)
exists_expression ::= EXISTS {collection_valued_path_expression | (subselect)}
subselect ::= SELECT [{ ALL | DISTINCT }] expression from_clause [where_clause]
    [group_by_clause] [having_clause]
group_by_clause ::= GROUP BY [single_valued_path_expression, ]*
    single_valued_path_expression
having_clause ::= HAVING conditional_expression
is_of_type_expression ::= identifier IS OF TYPE
    ([[ONLY] abstract_schema_name, ]* [ONLY] abstract_schema_name)
comparison_expression ::= expression comparison_operator { expression | ( subquery ) }
comparison_operator ::= = | > | >= | < | <= | <>

```



```

method ::= method_name( [[expression , ]* expression ] )
expression ::= term | expression {+|-} term
term ::= factor | term {*/|} factor
factor ::= {+|-} primary
primary ::= single_valued_path_expression | literal |
           ( expression ) | input_parameter | functions
functions ::=
    ABS(expression) |
    AVG([ALL|DISTINCT] expression) |
    BIGINT(expression) |
    CHAR({expression [, {ISO|USA|EUR|JIS}] } ) |
    CONCAT (expression , expression ) |
    COUNT({[ALL|DISTINCT] expression | *}) |
    DATE(expression) |
    DAY({expression} ) |
    DAYS( expression ) |
    DECIMAL( expression [,integer[,integer]])
    DIGITS( expression ) |
    DOUBLE( expression ) |
    FLOAT( expression ) |
    HOUR ( expression ) |
    INTEGER( expression ) |
    LCASE ( expression ) |
    LENGTH(expression) |
    LOCATE( expression, expression [, expression] ) |
    MAX([ALL|DISTINCT] expression) |
    MICROSECOND( expression ) |
    MIN([ALL|DISTINCT] expression) |
    MINUTE ( expression ) |
    MONTH( expression ) |
    REAL( expression ) |
    SECOND( expression ) |
    SMALLINT( expression ) |
    SQRT ( expression ) |
    SUBSTRING( expression, expression[, expression] ) |
    SUM([ALL|DISTINCT] expression) |
    TIME( expression ) |
    TIMESTAMP( expression ) |
    UCASE ( expression ) |
    YEAR( expression )

```

Comparison of EJB 2.0 specification and WebSphere query language

Item	EJB 2.0 specification	WebSphere Query	WebSphere Enterprise (Dynamic) Query
Bean methods	no	no	yes
Calendar comparisons	yes	yes	yes
Delimited identifiers	no	yes	yes
Dependent Value attributes	no	yes	yes
Dependent Value methods	no	no	yes
Dynamic Query APIs	no	no	yes
EXISTS predicate	no	yes	yes
Inheritance	no	yes	yes

Item	EJB 2.0 specification	WebSphere Query	WebSphere Enterprise (Dynamic) Query
Multiple element select clauses	no	no	yes
Order by	no	yes	yes
Scalar functions	yes *	yes	yes
Select clause	required	optional	optional
SQL Date/time expressions	no	yes	yes
String comparisons	= and <> only	= <> > <	= <> > <
Subqueries, aggregations, group by, and having clauses	no	yes	yes

* EJB 2.0 defines the following scalar functions: abs, sqrt, concat, length, locate and substring. WebSphere query and dynamic query support additional scalar functions as listed in the topic, EJB query: Scalar functions.

Chapter 11. Internationalizing applications

For your application to be used in multiple regions around the world, its user interfaces will need to support multiple locales and time zones. IBM WebSphere Application Server supports the maintenance and deployment of centralized message catalogs for the output of properly formatted, language-specific (*localized*) interface strings.

- If you are new to internationalization, read "Internationalization" before you continue.
 - For general information about internationalization, see "Resources for learning."
1. Identify localizable text in your application.
 2. Create the message catalogs necessary for the locales to be supported by your application.
 3. In your application code, compose the language-specific strings for output.
 4. Assemble your application code as one or more application components.
 5. Prepare the localizable-text package for deployment with your localized application. In this step, you create a deployment JAR.
 6. Assemble the application modules and the deployment JAR into a J2EE application.
 7. Deploy and manage the application.

Internationalization

An application that can present information to users according to regional cultural conventions is said to be *internationalized*: The application can be configured to interact with users from different localities in culturally appropriate ways. In an internationalized application, a user in one region sees error messages, output, and interface elements in the requested language. Date and time formats, as well as currencies, are presented appropriately for users in the specified region. A user in another region sees output in the conventional language or format for that region.

Historically, the creation of internationalized applications has been restricted to large corporations writing complex systems. Internationalization techniques have traditionally been expensive and difficult to implement, so they have been applied only to major development efforts. However, given the rise in distributed computing and in the use of the World Wide Web, application developers have been pressured to internationalize a much wider variety of applications. This requires making internationalization techniques much more accessible to application developers.

In an application that is not internationalized, the interface that the user sees is unalterably written into the application code. On the other hand, localizing the displayed strings adds a layer of abstraction into the design of the application. Instead of simply printing an error message, an internationalized application represents the error message with some language-neutral information; in the simplest case, each error condition corresponds to a key. To print a usable error message, then, the application looks up the key in the configured message catalog. Each message catalog is a list of keys with associated strings. Different message catalogs provide strings for the different languages supported. The application looks up the key in the appropriate catalog, retrieves the corresponding error message in the requested language, and prints this string for the user.

Localization of text can be used for far more than translating error messages. For example, by using keys to represent each element in a graphical user interface (GUI) and by providing the appropriate message catalogs, the GUI itself (buttons, menus, and so on) can support multiple languages. Extending support to additional languages requires that you provide message catalogs for those languages; in many cases, the application itself needs no further modification.

Internationalization of an application is driven by two variables, the time zone and the locale. The time zone indicates how to compute the local time as an offset from a standard time like Greenwich Mean Time. The locale is a collection of information about language, currency, and the conventions for presenting information like dates. In a localized application, the locale also indicates the message catalog from which an application is to retrieve message strings. A time zone can cover many locales, and a single locale can span time zones. With both time zone and locale, the date, time, currency, and language for users in a specific region can be determined.

Identifying localizable text

1. Determine which elements of the application need to be translated. Good candidates for localization include the following:
 - Graphical user interfaces: window titles, menus and menu items, buttons, on-screen instructions
 - Prompts in command-line interfaces
 - Application output: messages and logs
2. Assign a unique key to each element for use in message catalogs for the application. The key provides a language-neutral link between the application and language-specific strings in the message catalogs. Establishing a naming convention for keys before creating the catalogs can make writing code with these keys much more intuitive for interface programmers.

Suppose you are localizing the GUI for a banking system, and the first window contains a pull-down list to be used for selecting a type of account. The labels for the list and the account types in the list are good choices for localization. Three elements require keys: the list itself and two items in the list.

Create message catalogs for the language-specific strings.

Creating message catalogs

Identify strings that need to be localized.

You can create a catalog as either a subclass of `java.util.ResourceBundle` or a Java properties file. The properties-file approach is more common, because properties files can be prepared by people without programming experience and swapped in without modifying the application code.

1. For each string identified for localization, add a line to the message catalog that lists the string's key and value in the current language. In a properties file, each line has the following structure:
key = string associated with the key
2. Save the catalog, giving it a locale-specific name. To enable resolution to a specific properties file, the Java API specifies naming conventions for the properties files in a resource bundle as `bundleName_localeID.properties`. Give the

set of message catalogs a collective name, for example, `BankingResources`. For information about locale IDs that are recognized by the Java APIs, see "Resources for learning."

The following English catalog (`BankingResources_en.properties`) supports the labels for the list and its two list items:

```
accountString = Accounts
savingsString = Savings
checkingString = Checking
```

The corresponding German catalog (`BankingResources_de.properties`) supports the labels as follows:

```
accountString = Konten
savingsString = Sparkonto
checkingString = Girokonto
```

Write code to compose the language-specific strings.

Composing language-specific strings

Create message catalogs for the language-specific strings.

1. In application code, create a `LocalizableTextFormatter` instance, passing in required localization values.
2. Set other localization values as needed for more complex situations.
3. Generate a properly formatted, language-specific string.

When the application is finished, deploy your application. For more information, see "Preparing the localizable-text package for deployment" on page 458.

Localization API support

The package `com.ibm.websphere.i18n.localizabletext` contains classes and interfaces for localizing text. This package makes extensive use of the internationalization features of the standard Java APIs from Sun Microsystems, including the following:

- `java.util.Locale`
- `java.util.TimeZone`
- `java.util.ResourceBundle`
- `java.text.MessageFormat`

For more information about the standard Java APIs, see "Resources for learning."

The WebSphere localizable-text package wraps the Java support and extends it for efficient and simple use in a distributed environment. The primary class used by application programmers is `LocalizableTextFormatter`. Instances of this class are usually created in server programs, but client programs can also create them. Formatter instances are created for specific resource-bundle names and keys. Client programs that receive a `LocalizableTextFormatter` instance call its `format` method. This method uses the locale of the client application to retrieve the appropriate resource bundle and compose a locale-specific message based on the key.

For example, suppose that a distributed application supports both French and English locales; the server is using an English locale and the client, a French locale. The server creates two resource bundles, one each for English and French. When the client makes a request that triggers a message, the server creates a

LocalizableTextFormatter instance that contains the name of the resource bundle and the key for the message and passes the instance back to the client.

When the client receives the LocalizableTextFormatter instance, it calls the object's format method. By using the locale and name of the resource bundle, the format method determines the name of the resource bundle that supports the French locale and retrieves the message that corresponds to the key from the French resource bundle. Formatting of the message is transparent to the client.

In this simple example, the resource bundles reside centrally with the server. They do not have to exist with the client. Part of what the localizable-text package provides is the infrastructure to support centralized catalogs. This implementation uses an enterprise bean (a stateless session bean provided with the localizable-text package) to access the message catalogs. When the client calls the format method on the LocalizableTextFormatter instance, the following events occur:

1. The client application sets the time-zone and locale values in the LocalizableTextFormatter instance, either by passing them explicitly or through default values.
2. A call, LocalizableTextFormatterEJBFinder, is made to retrieve a reference to the formatter bean.
3. Information from the LocalizableTextFormatter instance, including the client's time zone and locale, is sent to the formatting bean.
4. The formatting bean uses the name of the resource bundle, the message key, the time zone, and the locale to compose a language-specific message.
5. The formatter bean returns the formatted message to the client.
6. The formatted message is inserted into the LocalizableTextFormatter instance and returned by the format method.

A call to the format method requires at most one remote call, to contact the formatter bean. As an alternative, the LocalizableTextFormatter instance can cache formatted messages, eliminating the remote call for subsequent uses. In addition, you can set a fallback string so that the application can return a readable string even if it cannot access the appropriate message catalog.

The resource bundles can be stored locally. The localizable-text package provides a static variable that indicates whether the bundles are stored locally (LocalizableConfiguration.LOCAL) or remotely (LocalizableConfiguration.REMOTE). However, the setting of this variable applies to all applications running within the same Java virtual machine.

LocalizableTextFormatter class

The LocalizableTextFormatter class, found in the package `com.ibm.websphere.i18n.localizabletext`, is the primary programming interface for using the localizable-text package. Instances of this class contain the information needed to create language-specific strings from keys and resource bundles.

LocalizableTextFormatter extends `java.lang.Object` and implements the following interfaces:

- `java.io.Serializable`
- `com.ibm.websphere.i18n.localizabletext.LocalizableText`
- `com.ibm.websphere.i18n.localizabletext.LocalizableTextL`
- `com.ibm.websphere.i18n.localizabletext.LocalizableTextTZ`
- `com.ibm.websphere.i18n.localizabletext.LocalizableTextLTZ`

Creation and initialization of class instances

LocalizableTextFormatter supports the following constructors:

- LocalizableTextFormatter()
- LocalizableTextFormatter(String resourceBundleName, String patternKey, String appName)
- LocalizableTextFormatter(String resourceBundleName, String patternKey, String appName, Object[] args)

The LocalizableTextFormatter instance must have certain values, such as resource-bundle name, key, and the name of the formatting application. If you do not pass these values in by using the second constructor listed previously, you can set them separately by making the following calls:

- setResourceBundleName(String resourceBundleName)
- setPatternKey(String patternKey)
- setApplicationName(String appName)

You can use a fourth method, setArguments(Object[] args), to set optional localization values after construction. See Processing of application-specific values at the end of this article. For a usage example, see "Composing complex strings."

API for formatting text

The formatting methods in LocalizableTextFormatter generate a string from a set of message keys and resource bundles, based on some combination of locale and time-zone values. Each method corresponds to one of the four localizable-text interfaces implemented. The following list indicates the interface in which each formatting method is defined:

- LocalizableText.format()
- LocalizableTextL.format(java.util.Locale locale)
- LocalizableTextTZ.format(java.util.TimeZone timeZone)
- LocalizableTextLTZ.format(java.util.Locale locale, java.util.TimeZone timeZone)

The format method with no arguments uses the locale and time-zone values set as defaults for the Java virtual machine. All four methods throw LocalizableException objects.

Location of message catalogs and the appName value

Applications written with the localizable-text package can access message catalogs locally or remotely. In a distributed environment, use of remote, centrally located message catalogs is appropriate. All clients can use the same catalogs, and maintenance of the catalogs is simplified. Local formatting is useful in test situations and appropriate under some circumstances. In order to support either local or remote formatting, a LocalizableTextFormatter instance must indicate the name of the formatting application. For example, when an application formats a message by using remote catalogs, the message is actually formatted by an enterprise bean on the server. Although the localizable-text package contains the code to automate the lookup of the formatter bean and the issue of a call to it, the application needs to know the name of the formatter bean. Several methods in the LocalizableTextFormatter class use a value described as *appName*; this refers to the name of the formatting application, which is not necessarily the name of the application in which the value is set.

Caching of messages

LocalizableTextFormatter can optionally cache formatted messages so that they do not have to be reformatted when needed again. By default, caching is not enabled,

but use `LocalizableTextFormatter.setCacheSetting(true)` to enable caching. When caching is enabled and the `format` method is called, the method determines whether the message has already been formatted. If so, the cached message is returned. If the message is not found in the cache, the message is formatted and returned to the caller, and a copy of the message is cached for future use.

If caching is disabled after messages have been cached, those messages remain in the cache until the cache is cleared by a call to `LocalizableTextformatter.clearCache()`. You can clear the cache at any time; the cache is automatically cleared when any of the following methods is called:

- `setResourceBundleName(String resourceBundleName)`
- `setPatternKey(String patternKey)`
- `setApplicationName(String appName)`
- `setArguments(Object[] args)`

API for providing fallback information

Under some circumstances, it can be impossible to format a message. The `localizable-text` package implements a fallback strategy, making it possible to get some information even if a message cannot be formatted correctly into the requested language. The `LocalizableTextFormatter` instance can optionally store fallback values for a message string, the time zone, and the locale. These can be ignored unless the `LocalizableTextFormatter` instance throws an exception. To set fallback values, call the following methods as appropriate:

- `setFallBackString(String message)`
- `setFallBackLocale(Locale locale)`
- `setFallBackTimeZone(TimeZone timeZone)`

For a usage example, see "Generating localized text."

Processing of application-specific values

The `localizable-text` package provides native support for localization based on time zone and locale, but one can construct messages on the basis of other values as well. If you need to consider variables other than locale and time zone in formatting localized text, write your own formatter class.

Your formatter class can extend `LocalizableTextFormatter` or independently implement some or all of the same `localizable-text` interfaces. As a minimum, your class must implement `java.io.Serializable` and at least one of the `localizable-text` interfaces and its corresponding `format` method. If your class implements more than one `localizable-text` interface and `format` method, the order of evaluation of the interfaces is as follows:

1. `LocalizableTextLTZ`
2. `LocalizableTextL`
3. `LocalizableTextTZ`
4. `LocalizableText`

As an example, the `localizable-text` package provides a class that reports the time and date (`LocalizableTextDateTimeArgument`). In that class, date and time formatting is localized in accordance with three values: locale, time zone, and style.

Creating a formatter instance

Server programs typically create `LocalizableTextFormatter` instances that are sent to clients as the result of some operation; clients format the objects at the appropriate time. Less typically, client programs create `LocalizableTextFormatter` objects locally.

1. If needed for your application, write your own formatter class. For more information about implementation, see "LocalizableTextFormatter class."
2. In application code, call the appropriate constructor for the formatter class and set required localization values. Some localization values, such as resource bundle name, key and formatting application, must be set, either through a constructor or soon after construction. Other localization values can be set only as needed. For more information about the API, see the related reference.

The following code creates a `LocalizableTextFormatter` instance by using the default constructor and then sets the required localization values:

```
import com.ibm.websphere.i18n.localizabletext.LocalizableException;
import com.ibm.websphere.i18n.localizabletext.LocalizableTextFormatter;
import java.util.Locale;

public void drawAccountNumberGUI(String accountType) {
    ...
    LocalizableTextFormatter ltf = new LocalizableTextFormatter();
    ltf.setPatternKey("accountNumber");
    ltf.setResourceBundleName("BankingSample.BankingResources");
    ltf.setApplicationName("BankingSample");
    ...
}
```

The application that is requesting a localized message can specify the locale and time zone for which the message is to be formatted, or the application can use the default values set for the Java virtual machine.

For example, a GUI can enable users to select the language in which to display the interface. A default value must be set initially so that the GUI can be created properly when the application first starts, but users can then change the locale for the GUI to suit their needs. The following code shows how to change the locale used by an application based on the selection of a menu item:

```
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
...
import java.util.Locale;

public void actionPerformed(ActionEvent event) {
    String action = event.getActionCommand();
    ...
    if (action.equals("en_us")) {
        applicationLocale = new Locale("en", "US");
        ...
    }
    if (action.equals("de_de")) {
        applicationLocale = new Locale("de", "DE");
        ...
    }
    if (action.equals("fr_fr")) {
        applicationLocale = new Locale("fr", "FR");
        ...
    }
    ...
}
```

For more information, see "Generating localized text."

Set optional localization values.

Setting optional localization values

In addition to setting localization values that are required by `LocalizableTextFormatter`, you can set a number of optional values in application code, either through the constructor or by calling any of several methods for that purpose. With optional values, you can do the following:

- Compose complex strings from variable substrings
 - Customize the formatting of strings, taking into account variables other than time zone and locale
1. In application code, add the optional values into an array of type `Object`.

```
Object[] arg = {new String(getAccountNumber())};
```
 2. Pass the array into a `LocalizableTextFormatter` instance. You can pass the array through the appropriate constructor or by calling the `setArguments(Object[])` method. For a usage example, see "Composing complex strings."

Note: Because the array is passed by value rather than by reference, any updates to the array variable after this point are not reflected in the `LocalizableTextFormatter` instance unless it is reset by calling the `setArguments(Object[])` method.

Write code to generate the localized text.

Composing complex strings

Identify strings that need to be localized.

The localized-text package supports the substitution of variable substrings into a localized string that is retrieved from the message catalog by key.

1. In the message catalog, specify the location of the substitution in the string to be retrieved by key. Variable components are designated by curly braces (for example, `{0}`).
2. In application code, create a `LocalizableTextFormatter` instance, passing in an array that contains the variable value. If the variable substring must itself be localized, you can create a nested `LocalizableTextFormatter` instance for it and pass the instance in instead of a value.
3. Generate a localized string. When a format method is called on a formatter instance, the formatter takes each element of the array passed in the previous step and substitutes it for the placeholder with the matching index in the string retrieved from the message catalog. For example, the value at index 0 in the array replaces the `{0}` variable in the retrieved string.

The following line from an English message catalog shows a string with a single substitution:

```
successfulTransaction = The operation on account {0} was successful.
```

The same key in message catalogs for other languages has a translation of this string with the variable at the appropriate location for each language.

The following code shows the creation of a single-element argument array and the creation and use of a `LocalizableTextFormatter` instance:

```

public void updateAccount(String transactionType) {
    ...
    Object[] arg = {new String(this.accountNumber)};
    ...
    LocalizableTextFormatter successLTF =
        new LocalizableTextFormatter ("BankingResources",
            "successfulTransaction",
            "BankingSample",
            arg);
    ...
    successLTF.format(this.applicationLocale);
    ...
}

```

Nesting formatter instances for localized substrings:

Identify strings that need to be localized.

The ability to substitute variable substrings into the strings retrieved from message catalogs adds a level of flexibility to the localizable-text package, but this capability is of limited use unless the variable value itself can be localized. You can do this by nesting `LocalizableTextFormatter` instances.

1. In the message catalog, add entries that correspond to potential values for the variable substring.
2. In application code, create a `LocalizableTextFormatter` instance for the variable substring, setting required localization values.
3. Create a `LocalizableTextFormatter` instance for the primary string, passing in an array that contains the formatter instance for the variable substring.

The following line from an English message catalog shows a string entry with two substitutions and entries to support the localizable variable at index 0 (the second variable in the string, the account number, does not need to be localized):

```

successfulTransaction = The {0} operation on account {1} was successful.
depositOpString = deposit
withdrawOpString = withdraw

```

The following code shows the creation of the nested formatter instance and its insertion (with the account number variable) into the primary formatter instance:

```

public void updateAccount(String transactionType) {
    ...
    // Successful deposit
    LocalizableTextFormatter opLTF =
        new LocalizableTextFormatter("BankingResources",
            "depositOpString",
            "BankingSample");
    Object[] args = {opLTF, new String(this.accountNumber)};
    ...
    LocalizableTextFormatter successLTF =
        new LocalizableTextFormatter ("BankingResources",
            "successfulTransaction",
            "BankingSample",
            args);
    ...
    successLTF.format(this.applicationLocale);
    ...
}

```

Generating localized text

Create a formatter instance and set localization values as needed.

1. If needed, customize the formatting behavior.
2. In application code, call the appropriate format method.

You can provide fallback behavior for use if the appropriate message catalog is not available at formatting time.

The following code generates a localized string. If the formatting fails, the application retrieves and uses a fallback string instead of the localized string:

```
import com.ibm.websphere.i18n.localizabletext.LocalizableException;
import com.ibm.websphere.i18n.localizabletext.LocalizableTextFormatter;
import java.util.Locale;

public void drawAccountNumberGUI(String accountType){
    ...
    LocalizableTextFormatter ltf = new LocalizableTextFormatter();
    ...
    ltf.setFallbackString("Enter account number: ");
    try {
        msg = new Label(ltf.format(this.applicationLocale), Label.CENTER);
    }
    catch (LocalizableException le) {
        msg = new Label(ltf.getFallbackString(), Label.CENTER);
    }
    ...
}
```

When the application is finished, deploy your application. For more information, see "Preparing the localizable-text package for deployment."

Customizing the behavior of a formatting method

You can customize formatting behavior by passing your own formatter classes into a `LocalizableTextFormatter` instance through an array of optional values. This enables you to take variables other than locale and time zone into account when formatting localized text.

1. Write your own formatter class. For more information about implementation, see "LocalizableTextFormatter class."
2. In application code, create an instance of your formatter class as appropriate and pass it with any other optional localization values into an instance of `LocalizableTextFormatter`. When the `LocalizableTextFormatter` instance reads the instance that has been passed in, it attempts to call `format()` on the passed-in instance. The string returned is then processed with any other elements in the array.

The localizable-text package provides an example of a user-defined class, called `LocalizableTextDateTimeArgument`. This class enables date and time information to be selectively formatted according to the style values defined in `java.text.DateFormat` as well as constants defined within `LocalizableTextDateTimeArgument` itself.

Preparing the localizable-text package for deployment

Write code to compose the language-specific strings.

The `LocalizableTextEJBDeploy` tool is used to create a deployment JAR for the Localizable Text service. You must deploy the enterprise bean in each enterprise application that requires support for localized text.

1. Make sure the **LocalizableTextEJBDeploy** tool (`ltext.jar`) exists in the `lib` directory under the product's installation root directory.
2. Set up a working directory for the **LocalizableTextEJBDeploy** tool to use. You will need to pass this location to the tool through a command-line interface.
3. Run the **LocalizableTextEJBDeploy** tool. You might be asked if you want to regenerate deployment code for the LocalizableText bean. Do not redeploy the bean; if you do, the generated JNDI name will be incorrect.

To deploy the bean on multiple hosts and servers, run the tool for each host/server combination. This generates a unique JNDI name for each deployment. After the tool is run, a deployment JAR is located in the working directory you specified.

Assemble the deployment JAR in an enterprise application with other application components. As part of preparing for deployment, verify the following:

- Add the resource bundles for your application to the EAR as files.
- Add the location of the EAR to the server's class path. This is so the resource bundles can be located on the virtual host and server.

The same deployment JAR can be included in several enterprise applications.

LocalizableTextEJBDeploy command

This topic describes the command-line syntax for the **LocalizableTextEJBDeploy** tool. The file that contains this tool (`ltext.jar`) must be located in the `lib` directory of the product installation root.

```
LocalizableTextEJBDeploy
-a applicationName
-h virtualHostName
-i installationDirectory
-s serverName
-w workingDirectory
```

Parameters

The required parameters, which can be specified in any order, follow:

applicationName

The name of the formatting session bean. This name is used in `LocalizableTextFormatter` instances to specify where the actual formatting takes place. If the name cannot be resolved at run time, the `format` method throws an exception.

virtualHostName

The name of the virtual host on which the formatting session bean is deployed. This value is case-sensitive on all operating platforms.

installationDirectory

The location at which the application server product is installed.

serverName

The name of the application server. If this argument is not specified, the default server name for the product is used.

workingDirectory

A location for the tool to use temporarily.

Internationalization: Resources for learning

Use the following links to find relevant supplemental information about internationalization. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to this product but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

View links to additional information about:

- Programming instructions and examples
- Programming specifications

Programming instructions and examples

- Java internationalization tutorial

An online tutorial that explains how to use the Java 2 SDK Internationalization API.

Programming specifications

- Java 2 SDK, Standard Edition Documentation: Internationalization

The Java internationalization documentation from Sun Microsystems, including a list of supported locales and encodings.

- Making the WWW truly World Wide

The W3C's effort to make World Wide Web technology work with the many writing systems, languages, and cultural conventions of the global community:

- developerWorks - Unicode

Articles on various subjects relating to Unicode, from IBM's developerWorks.

Chapter 12. Using the transaction service

These topics provide information about using transactions with WebSphere applications

WebSphere applications can use transactions to coordinate multiple updates to resources as atomic units (as indivisible units of work) such that all or none of the updates are made permanent.

In WebSphere Application Server, transactions are handled by three main components:

- A transaction manager that 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 transaction manager that 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. In addition, WebSphere Application Server for z/OS V5 supports the coordination of resource managers through RRS (z/OS resource recovery services).
- A container in which the J2EE application runs. The container manages the enlistment of XAResources on behalf of the application when the application performs updates to transactional resource managers (for example, databases). Optionally, the container can control the demarcation of transactions for enterprise beans configured for container-managed transactions.
- An application programming interface (UserTransaction) that is available to bean-managed enterprise beans and servlets. This allows such application components to control the demarcation of their own transactions.

For more information about using transactions with WebSphere applications, see the following topics:

- Transaction support in WebSphere Application Server
- Developing a WebSphere application to use transactions
- Classifying WebSphere transaction workload for WLM
- Configuring transaction properties for an application server
- Using local transactions
- Setting transactional attributes in the deployment descriptor
- Using bean-managed transactions
- Managing active transactions
- Troubleshooting transactions
- Transaction service exceptions
- UserTransaction interface - methods available

Transaction support in WebSphere Application Server

A transaction is unit of activity within which multiple updates to resources can be made atomic (as an indivisible unit of work) such that all or none of the updates are made permanent. For example, multiple SQL statements to a relational database are committed atomically by the database during the processing of an SQL COMMIT statement. In this case, the transaction is contained entirely within the database manager and can be thought of as a resource manager local transaction (RMLT). In some contexts, a transaction is referred to as a logical unit of work (LUW).

The way that applications use transactions depends on the type of application component, as follows:

- 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 (servlets) use bean-managed transactions.

WebSphere Application Server is a transaction manager that supports the coordination of resource managers through their XAResource interface and participates in distributed global transactions with other OTS 1.2 compliant transaction managers (for example J2EE 1.3 application servers). WebSphere applications can also be configured to interact with databases, JMS queues, and JCA connectors through their local transaction support when distributed transaction coordination is not required.

In addition to supporting the coordination of XAResource-based resource managers, WebSphere Application Server for z/OS V5 supports the coordination of resource managers through RRS (z/OS resource recovery services). RRS-compliant resource managers include DB2, MQSeries, IMS, and CICS. IBM WebSphere Application Server for z/OS is capable of coordinating a mix of RRSTransactional resource managers and XA capable resource managers under the same global transaction.

Resource managers that offer transaction support can be categorized into those that support two-phase coordination (by offering an XAResource interface) and those that support only one-phase coordination (for example through a LocalTransaction interface). The WebSphere Application Server transaction support provides coordination, within a transaction, for any number of two-phase capable resource managers. It also enables a single one-phase capable resource manager to be used within a transaction in the absence of any other resource managers, although a WebSphere transaction is not necessary in this case.

Resource managers that offer transaction support can be categorized into those that support two-phase coordination (by offering an XAResource interface or by supporting RRS) and those that support only one-phase coordination (for example through a LocalTransaction interface). The WebSphere Application Server transaction support provides coordination, within a transaction, for any number of two-phase capable resource managers. It also enables a single one-phase capable resource manager to be used within a transaction in the absence of any other resource managers, although a WebSphere transaction is not necessary in this case.

You can use transaction classes to classify client workload for workload management. The workload is different WebSphere transactions targeted to separate servant regions, each with goals defined by appropriate service classes. Each transaction is dispatched in its own WLM enclave in a servant region process, and is managed according to the goals of its service class. The server controller, which workload management views as a queue manager, uses the enclave associated with a client request to manage the priority of the work. If the work has a high priority, workload management can direct the work to a high-priority servant in the server. If the work has a low priority, workload management can direct the work to a low-priority servant. The effect is to partition the work according to priority within the same server.

With the Last Participant Support of WebSphere Application Server Enterprise, you can coordinate the use of a single one-phase commit (IPC) capable resource with

any number of two-phase commit (2PC) capable resources in the same global transaction. At transaction commit, the two-phase commit resources are prepared first using the two-phase commit protocol, and if this is successful the one-phase commit-resource is then called to commit(one_phase). The two-phase commit resources are then committed or rolled back depending on the response of the one-phase commit resource.

The ActivitySession service of WebSphere Application Server Enterprise provides an alternative unit-of-work (UOW) scope to that provided by global transaction contexts. It is a distributed context that can be used to coordinate multiple one-phase resource managers. The WebSphere EJB container and deployment tooling support ActivitySessions as an extension to the J2EE programming model. EJBs can be deployed with lifecycles that are influenced by ActivitySession context, as an alternative to transaction context. An application can then interact with a resource manager through its LocalTransaction interface for the period of a client-scoped ActivitySession rather than just the duration of an EJB method.

Resource manager local transaction (RMLT)

A resource manager local transaction (RMLT) is a resource manager's view of a local transaction; that is, it represents a unit of recovery on a single connection that is managed by the resource manager.

Resource managers include:

- Enterprise Information Systems that are accessed through a resource adapter, as described in the J2EE Connector Architecture 1.0.
- Relational databases that are accessed through a JDBC datasource.
- JMS queue and topic destinations.

Resource managers offer specific interfaces to enable control of their RMLTs. J2EE connector resource adapters that include support for local transactions provide a LocalTransaction interface to enable applications to request that the resource adapter commit or rollback RMLTs. JDBC datasources provide a Connection interface for the same purpose.

The boundary at which all RMLTs must be complete is defined in WebSphere Application Server by a local transaction containment (LTC).

Global transactions

If an application uses two or more resources, then an external transaction manager is needed to coordinate the updates to both resource managers in a global transaction.

Global transaction support is available to web and enterprise bean J2EE components. Enterprise bean components can be subdivided into beans that exploit container-managed transactions (CMT) or bean-managed transactions (BMT).

BMT enterprise beans and web components can use the Java Transaction API (JTA) UserTransaction interface to define the demarcation of a global transaction. The UserTransaction interface is obtained by a JNDI lookup of `java:comp/UserTransaction`. The UserTransaction is not available to the following components:

- CMT enterprise beans. Any attempt by such beans to obtain the interface results in an exception in accordance with the EJB specification.
- Client applications running outside the Web and EJB containers.

Ensure that programs that perform a JNDI lookup of the UserTransaction interface, use an InitialContext that resolves to a local implementation of the interface. Also ensure that such programs use a JNDI location appropriate for the EJB version.

Before the EJB 1.1 specification, the JNDI location of the UserTransaction interface was not specified. Each EJB container implementor defined it in an implementation-specific manner. Earlier versions of WebSphere Application Server, up to and including Version 3.5.x (without EJB 1.1), bind the UserTransaction interface to a JNDI location of jta/usertransaction. WebSphere Application Server Version 4, and later releases, bind the UserTransaction interface at the location defined by EJB 1.1, which is java:comp/UserTransaction. WebSphere Application Server, Version 5 no longer provides the jta/usertransaction binding within Web and EJB containers to applications at a J2EE level of 1.3 or later. For example, EJB 2.0 applications can use only the java:comp/UserTransaction location.

Local transaction containment (LTC)

A local transaction containment (LTC) is used to define the application server behavior in an unspecified transaction context.

(Unspecified transaction context is defined in the Enterprise JavaBeans 2.0 Specification.)

A LTC is a bounded unit-of-work scope within which zero, one, or more resource manager local transactions (RMLTs) can be accessed. The LTC defines the boundary at which all RMLTs must be complete; any incomplete RMLTs are resolved, according to policy, by the container. An LTC is local to a bean instance; it is not shared across beans even if those beans are managed by the same container. LTCs are started by the container before dispatching a method on a J2EE component (such as an enterprise bean or servlet) whenever the dispatch occurs in the absence of a global transaction context. LTCs are completed by the container depending on the application-configured LTC boundary; for example at the end of the method dispatch. There is no programmatic interface to the LTC support; rather LTCs are managed exclusively by the container and configured by the application deployer through transaction attributes in the application deployment descriptor.

A local transaction containment cannot exist concurrently with a global transaction. If application component dispatch occurs in the absence of a global transaction, the container always establishes an LTC for J2EE components at J2EE 1.3 or later. The only exceptions to this are as follows:

- Where application component dispatch occurs without container interposition; for example, for a stateless session bean create or a servlet-initiated thread.
- J2EE 1.2 web components.
- J2EE 1.2 BMT enterprise beans.

A local transaction containment cannot exist concurrently with a global transaction. If application component dispatch occurs in the absence of a global transaction, the container always establishes an LTC. The only exceptions to this behavior is when an application component dispatch occurs without container interposition; for example, for a stateless session bean create.

Local and global transaction considerations

Applications use resources, such as JDBC data sources or connection factories, that are configured through the Resources view of the WebSphere Application Server Administrative Console. How these resources participate in a global transaction depends on the underlying transaction support of the resource provider. For

example, most JDBC providers can provide either XA or non-XA versions of a data source. A non-XA data source can support only resource manager local transactions (RMLTs), but an XA data source can support two-phase commit coordination, as well as local transactions.

Additionally, some JDBC Providers such as the DB2 for z/OS Local JDBC Provider support the use of z/OS Resource Recovery Service (RRS) to coordinate transaction processing. This type of JDBC Provider is RRSTransactional. When RRS is used, both local and global transactions are supported.

If an application uses two or more resource providers that support only RMLTs, then atomicity cannot be assured because of the one-phase nature of these resources. To ensure atomic behavior, the application should use resources that support XA coordination and should access them within a global transaction.

If an application uses two or more resource providers that support only RMLTs, then atomicity cannot be assured because of the one-phase nature of these resources. To ensure atomic behavior, the application should use resources that support XA coordination or RRS coordination and should access them within a global transaction.

If an application uses only one RMLT, the atomic behavior can be guaranteed by the resource manager, which can be accessed under a local transaction containment context.

An application can also access a single resource manager under a global transaction context, even if that resource manager does not support the XA coordination. An application can do this, because WebSphere Application Server performs an “only resource optimization” and interacts with the resource manager under a RMLT. Within a global transaction context, any attempt to use more than one resource provider that supports only RMLTs causes the global transaction to be rolled back.

At any moment, an instance of an enterprise bean can have work outstanding in either a global transaction context or a local transaction containment context, but never both. An instance of an enterprise bean can change from running under one type of context to the other (in either direction), if all outstanding work in the original context is complete. Any violation of this principle causes an exception to be thrown when the enterprise bean tries to start the new context.

Developing components to use transactions

These topics provide information about developing WebSphere application components to use transactions

The way that applications use transactions depends on the type of application component, as follows:

- 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 (servlets) use bean-managed transactions.

You configure whether a component uses container- or bean-managed transactions by setting an appropriate value on the Transaction type deployment attribute, as

described in Setting transactional attributes in the deployment descriptor. You can also configure other transactional deployment descriptor attributes.

If you want a session bean to manage its own transactions, you must write the code that explicitly demarcates the boundaries of a transaction as described in Using bean-managed transactions.

Similarly, if you want a Web component to use transactions, you must write the code that explicitly demarcates the boundaries of a transaction as described in Using bean-managed transactions.

Setting transactional attributes in the deployment descriptor

Use this task to configure the transactional deployment descriptor attributes associated with an EJB or Web module, to enable a J2EE application to use transactions.

To set transactional attributes in the deployment descriptor for an application component (enterprise bean or servlet), complete the following steps:

1. Start the Application Assembly Tool.
2. Create or edit the application EAR file. For example, to change attributes of an existing application, click **File-> Open** then select the EAR file.
3. In the navigation pane, select the component instance; for example:
 - For a session bean, expand **ejb_module_instance-> Session beans** then select the bean instance.
 - For a servlet, expand **web_application-> Web Components** then select the servlet instance.

A property dialog notebook for the component is displayed in the property pane.

4. In the property pane, select the Advanced tab.
5. Set the **Transaction type** attribute, which defines the transactional manner in which the container invokes a method. You can set this attribute to Container or Bean, as follows:
 - For a session bean to use container-managed transactions, set Container
 - For a session bean to use bean-managed transactions, set Bean
 - For an entity bean, set Container
 - For a Web component (servlet), set Bean
6. In the property pane, select the IBM Extensions tab.
7. Configure J2EE component extensions attributes for extended local transaction containment. To enable management of local transaction containments, configure the following EJB extensions attributes. These attributes configure, for the component, the behaviour of the container's local transaction containment (LTC) environment that the container establishes whenever a global transaction is not present.

Boundary

Specifies the duration of a local transaction context. You can set this attribute to **Bean method** or **ActivitySession**, as described in Entity bean assembly settings.

Note: The ActivitySession option is not supported in the web container.

This property can be changed on WAS Enterprise only.

Boundary

Specifies the duration of a local transaction context. You can set this attribute to either **Bean method**, as described in Entity bean assembly settings.

Resolution control

Specifies how the local transaction is to be resolved before the local transaction context ends: by the application through user code or by the EJB container. You can set this attribute to either **Application** or **ContainerAtBoundary**, as described in Entity bean assembly settings.

Unresolved action

Specifies the action that the container must take when the local transaction context scope ends, if resources are uncommitted by an application in a local transaction and the **Resolution control** is set to **Application**. You can set this attribute to either **Commit** or **Rollback**, as described in Entity bean assembly settings.

8. [For EJB components only] For container-managed transactions, configure how the container must manage the transaction boundaries when delegating a method invocation to an enterprise bean's business method:
 - a. In the navigation pane, select **Container Transactions**. This displays a table of the methods for enterprise beans.
 - b. For each method of the enterprise bean set the **Transaction attribute** attribute to an appropriate value, as defined in Container transaction assembly settings.

Using bean-managed transactions

This topic describes how to enable a session bean or servlet to use bean-managed transactions, to manage its own transactions directly instead of letting the container manage the transactions.

Note: Entity beans cannot manage transactions (so cannot use bean-managed transactions).

To enable a session bean or servlet to use bean-managed transactions, complete the following steps:

1. Set the **Transaction type** attribute in the component's deployment descriptor to **Bean**, as described in Setting transactional attributes in the deployment descriptor.
2. Write the component code to actively manage transactions. When writing the code required by a component to manage its own transactions, remember the following basic rules:
 - An instance of a stateless session bean cannot reuse the same transaction context across multiple methods called by an EJB client.
 - An instance of a stateful session bean can reuse the same transaction context across multiple methods called by an EJB client.

The following code extract shows the standard code required to obtain an object encapsulating the transaction context. There are three basic steps involved:

- The component class must set the value of the `javax.ejb.SessionContext` object reference in the `setSessionContext` method.
- A `javax.transaction.UserTransaction` object is created by calling a lookup on `"java:comp/UserTransaction"`.
- The `UserTransaction` object is used to participate in the transaction by calling transaction methods such as `begin` and `commit` as needed. If an enterprise

bean begins a transaction, it must also complete that transaction either by invoking the commit method or the rollback method.

Code example: Getting an object that encapsulates a transaction context

```
...
import javax.transaction.*;
...
public class MyStatelessSessionBean implements SessionBean {
private SessionContext mySessionCtx =null;
...
public void setSessionContext (SessionContext ctx)throws EJBException {
mySessionCtx =ctx;
}
...
    public float doSomething(long arg1)throws FinderException,EJBException {
        UserTransaction userTran = (UserTransaction)initCtx.lookup(
            "java:comp/UserTransaction");
        ...
        //User userTran object to call transaction methods
        userTran.begin ();
        //Do transactional work
        ...
        userTran.commit ();
        ...
    }
    ...
}
```

Classifying WebSphere transaction workload for WLM

This topic describes how to use transaction classes to classify client workload for workload management. The workload is different WebSphere transactions targeted to separate servant regions, each with goals defined by appropriate service classes. Each transaction is dispatched in its own WLM enclave in a servant region process, and is managed according to the goals of its service class.

This topic describes steps to classify transaction workload as a way of managing the workload service objectives. You also need to define the service objectives (goals) for the service classes used. In addition, you must define the service objectives of the WebSphere for z/OS servers and your business application servers.

For more information about defining service objectives (goals) for each service class, see the *z/OS MVS Planning: Workload Management* book, SA22-7602, for example at <http://publibz.boulder.ibm.com/epubs/pdf/iea2w131.pdf>, or the z/OS WLM Web page at <http://www.ibm.com/servers/eserver/zseries/zos/wlm/>.

You can classify your WebSphere work using the WLM CB-type classification criteria:

- Server name (CN)
- Server instance name (SI)
- User ID assigned to the transaction (UI)
- Transaction class (TC)

Note: To get started, you do not need to define special classification rules and work qualifiers, but you may want to do this for your production system.

To classify work using server and userid criteria, you use a combination of the WLM Workload Classification rules in the WLM ISPF dialog panels. For more

information about defining WLM Classification rules, see the section “Workload management and WebSphere for z/OS” in the *Installation and Customization Guide*, which includes an example of classification rules.

To classify work using transaction classes, you define and use transaction class mappings, as described in this task. The steps to classify work using transaction classes are:

1. Define transaction class mappings based on the HTTP virtual host name, port number, and URI (Universal Resource Identifier - encoded address for any resource on the Web) provided with each work HTTP or HTTPS request.
 - a. Create a Transaction Class mapping file (as a simple text file). For example: `/wasconfig/t5was/MyTrMapFile.txt`
 - b. Edit the Transaction Class mapping file to define each transaction class mapping that you want to use. Define each mapping on a separate line, using the following syntax:

```
TransClassMap host:port uritemplate tclass
```

Note: In the host or port fields, you can use wildcard characters only for the entire field as shown in the following example.

This syntax is the same syntax as for WebSphere for z/OS Version 4.0.1. For more information about this syntax, see Transaction class mapping file entries. For example:

```
TransClassMap wsc4.washington.ibm.com:9080 /MyIVT/index.* TCLMYIVT
TransClassMap wsc4.washington.ibm.com:9080 /MyIVT/ivtejb TCLMYEJB
TransClassMap wsc4.washington.ibm.com:* /SuperSnoop* TCLSNOOP
TransClassMap wsc4.washington.ibm.com:* /ssb/* TCLSSB
TransClassMap *:* /admin* TCLADMIN
```

2. Specify the Transaction Class mapping file on the administrative properties for each server that is to handle work classified by transaction class. To specify the Transaction Class mapping file for a server, use the administrative console to complete the following steps:
 - a. In the navigation pane, click **Servers > Application Servers**.
 - b. In the content pane, select the server instance, *server_name*.
 - c. In the Additional Properties list in the contents pane, select **Web Container**.
 - d. In the Additional Properties list for the Web container, select **Advanced Settings**.
 - e. In the **Transaction Class Mapping** field, type the name of the Transaction Class mapping file that you edited in an earlier step. For example: `/wasconfig/t5was/MyTrMapFile.txt`

This sets the following variable in the server’s was.env file:

```
protocol_http_transport_class_mapping_file=/wasconfig/t5was/MyTrMapFile.txt
```

- f. If you want to use a transaction class to classify outbound data that is delivered in response to HTTP and HTTPS requests, select the TCLASS option in the **Network QoS** field. If you specify TCLASS, WebSphere for z/OS uses the transaction class value that was used to classify the inbound request to the z/OS Workload Manager.

The following table shows classification rules for CB-type work in which the default service class is WSMED and has a reporting class of RWSDEFLT. Work run in the WSPROD WebSphere server is classified as WSMED with a reporting class of RWSPROD, unless it has a transaction class of TCLASS1, TCLASS2, or TCLASS2 assigned through the transaction class mapping file below.

Qualifier #	Qualifier type	Qualifier name	Start position	Service Class	Report Class
			Default:	WSMED	RWSDEFLT
1	CN	WSPROD	1	WSMED	RWSPROD
2	. TC	. TCLASS1		WSFAST	RWSPRD1
2	. TC	. TCLASS2		WSMED	RWSPRD2
2	. TC	. TCLASS5		WSSLOW	RWSPRD5
1	CN	WSTEST	1	WSSLOW	RTSTEST
2	. UI	. USER1		WSMED	RTSTSTU2
2	. TC	. TCLASS5		WSSLOW	RTSTST5

The following table shows how work can be assigned a transaction class based on its host name, port number, or URI. For example, a web request of `http://ibm.com:80/Webap1/myservlet` handled by the WSPROD server would be assigned a transaction class of TCLASS1, a service class of WSFAST, and a reporting class of RWSPRD1 by the classification rules shown above.

```
TransClassMap www.ibm.com:80 /Webap1/myservlet TCLASS1
TransClassMap www.ibm.com:* /Webap1/myservlet TCLASS2
TransClassMap *:443 * TCLASS3
TransClassMap ** /Webap1/myservlet TCLASS4
TransClassMap www.ibm.com:* /Webap5/* TCLASS5
TransClassMap * * TCLASS6
```

Configuring transaction properties for an application server

Use this task to configure the transaction properties for an application server; for example, to define the location of the directory that contains the transaction log or to change default timeouts associated with transactions.

To configure the transaction properties for an application server, complete the following steps:

1. Start the Administrative console
2. In the navigation pane, select **Servers-> Manage Application Servers-> *your_app_server***. This displays the properties of the application server, *your_app_server*, in the content pane.
3. Select the Transaction Service tab, to display the properties page for the transaction service, as two notebook pages:

Configuration

The values of properties defined in the configuration file. If you change these properties, the new values are applied when the application server next starts.

Runtime

The runtime values of properties. If you change these properties, the new values are applied immediately, but are overwritten with the Configuration values when the application server next starts.

4. Select the Configuration tab, to display the transaction-related configuration properties.
5. If you want to change the directory in which transaction logs are written, type the full pathname of the directory in the **Transaction log directory** field. You can check the current runtime value of **Transaction log directory**, by clicking the Runtime tab.

You can also specify a size for the transaction logs, as described in the following step.

Note: If you change the transaction log directory, you should apply the change and restart the application server as soon as possible, to

minimize the risk of problems caused that might occur before the application server is restarted. For example, if a problem causes the server to fail (with in-flight transactions), the server next starts with the new log directory and is unable to automatically resolve in-flight transactions that were recorded in the old log directory.

6. If you want to change the default file size of transaction log files, modify the **Transaction log directory** field to include a file size setting, in the following format:

directory_name;file_size

Where

- *directory_name* is the name of the transaction log directory
- *file_size* is the new default size specified in bytes. The *nK* or *nM* suffix can be used to indicate kilobytes or megabytes. If you do not specify a file size value, the default value of 1M is used.

For example, `c:\tranlogs;2M` indicates the files are to be created with 2M bytes size and stored in the directory `c:\tranlogs`.

In a non-production environment, you can use the transaction log directory value of `;0` to disable transaction logging. (There must be no directory name element before the size element of 0.) You should not disable transaction logging in a production environment, because this prevents recovery after a system failure and, therefore, data integrity cannot be guaranteed.

7. In the **Total transaction lifetime timeout** field, type the number of seconds a transaction can remain inactive before it is ended by the transaction service. A value of 0 (zero) indicates that there is no timeout limit.
8. In the **Maximum transaction timeout** field, type the number of seconds for which a transaction started by or propagated into this application server may execute before it is ended by the transaction service. A value of 0 (zero) indicates that there is no timeout limit.
9. In the **Client inactivity timeout** field, type the number of seconds after which a client is considered inactive and the transaction service ends any transactions associated with that client. A value of 0 (zero) indicates that there is no timeout limit.
10. Click **OK**.
11. Stop then restart the application server.

If you change the transaction log directory configuration property to an incorrect directory name, the application server will restart but be unable to open the transaction logs. You should change the configuration property to a valid directory name, then restart the application server.

Transaction service settings

Use this page to modify transaction service settings.

To view this administrative console page, click **Servers > Application Servers > server > Transaction Service**.

Transaction log directory

Specifies the name of a directory for this server where the transaction service stores log files for recovery. Specifies the location of the JTA Partner Log.

A blank value in the server configuration is expanded by the transaction log at startup as the directory *(install_root)/tranlog/(server_name)*.

When the application running on the WebSphere product accesses more than one resource, the WebSphere product stores transaction information to properly coordinate and manage the distributed transaction. In a higher transaction load, this persistence slows down performance of the application server due to its dependency on the operating system and the underlying storage systems.

To achieve better performance, move the transaction log files to a storage device with more physical disk drives, or preferably RAID disk drives. When the log files are moved to the file systems on the RAIDed disks, the task of writing data to the physical media is shared across the multiple disk drives. This allows more concurrent access to persist transaction information and faster access to that data from the logs. Depending upon the design of the application and storage subsystem, performance gains can range from 10% to 100%, or even more in some cases.

This change is applicable only to the configuration where the application uses distributed resources or XA transactions, for example, multiple databases and resources are accessed within a single transaction. Consider setting this property when the application server shows one or more of following signs:

- CPU utilization remains low despite an increase in transactions
- Transactions fail with several time outs
- Transaction rollbacks occur with *unable to enlist transaction* exception
- Application server hangs in middle of a run and requires the server to be restarted
- The disk on which an application server is running shows higher utilization

Data type	String
Default	Initial value is the %WAS_HOME%/tranlog/(<i>server_name</i>) directory and a default size of 1MB.
Recommended	Create a file system with at least 3-4 disk drives RAIDed together in a RAID-0 configuration. Then, create the transaction log on this file system with the default size. When the server is running under load, check the disk input and output. If disk input and output time is more than 5%, consider adding more physical disks to lower the value. If disk input and output is low, but the server is still high, consider increasing the size of the log files.

This log is used for recovery of XA resources.

When the application that runs on the WebSphere product accesses XA resources, the WebSphere product stores information about the resource to enable XA transaction recovery.

Syntax

[location type URL tag] location specification

where

- *location type URL tag* specifies the optional location type for the JTA Partner Log:
 - *dir://* specifies that the JTA Partner Log location is in a fully qualified HFS directory specified by *location specification*. *dir://* is the default.
- *location specification* specifies the location name for the JTA Partner Log:

- If the *location type URL tag* is `dir://`, use a fully qualified HFS directory for the *location specification*. The complete name of the directory must be unique within the WebSphere node.

Default

`dir://install root/tranlog/server name`

For additional information, see the *WebSphere Application Server for z/OS V5.0 Installation and Customization* manual.

Total transaction lifetime timeout

Specifies the maximum duration, in seconds, for transactions on this application server. Specifies the maximum duration, in seconds, for container managed transactions started by this application server.

Component managed transactions that do not have a time-out explicitly set are also assigned this value.

Any transaction that is not requested to complete before this time-out is rolled back. If set to 0, there is no time-out limit.

Any transaction that is not requested to complete before this time-out is rolled back. If set to 0, only the maximum transaction timeout configuration value applies.

Data type	Integer
Units	Seconds
Default	120
Range	0 to 2 147 483 647
Range	0 to 2 147 040

Client inactivity timeout

Specifies the maximum duration, in seconds, between transactional requests from a remote client.

Any period of client inactivity that exceeds this timeout results in the transaction rolling back in this application server. If set to 0, there is no timeout limit.

Data type	Integer
Units	Seconds
Default	60
Range	0 to 2 147 483 647

Maximum Transaction Timeout

Specifies the maximum duration, in seconds, that transactions started by or propagated into this application server are allowed to execute.

This value limits the upper bound of all other transaction related time-outs. For example, assume a component attempts to set a transaction time-out of 360 seconds, and the Maximum Transaction Timeout setting is 300 seconds. The Maximum Transaction Time-out setting of 300 seconds is used.

Data type	Integer
Units	Seconds
Default	300
Range	0 to 2 147 040

Using local transactions

Local transaction containment (LTC) support, and its configuration through local transaction extended deployment descriptors, gives IBM WebSphere Application Server application programmers a number of advantages. This topic describes those advantages and how they relate to the settings of the local transaction extended deployment descriptors. This topic also describes points to consider to help you best configure transaction support for some example scenarios that use local transactions.

Develop an enterprise bean or servlet that accesses one or more databases that are independent and require no coordination.

If an enterprise bean does not need to use global transactions, it is often more efficient to deploy the bean with the Container Transaction deployment descriptor **Transaction** attribute set to Not supported instead of Required.

With the extended local transaction support of IBM WebSphere Application Server, applications can perform the same business logic in an unspecified transaction context as they can under a global transaction. An enterprise bean, for example, runs under an unspecified transaction context if it is deployed with a **Transaction** attribute of Not supported or Never.

The extended local transaction support provides a container-managed, implicit local transaction boundary within which application updates can be committed and their connections cleaned up by the container.

Applications can then be designed with a greater degree of independence from deployment concerns. This makes using a **Transaction** attribute of Supports much simpler, for example, when the business logic may be called either with or without a global transaction context.

An application can follow a get-use-close pattern of connection usage regardless of whether or not the application runs under a transaction. The application can depend on the close behaving in the same way and not causing a rollback to occur on the connection if there is no global transaction.

There are many scenarios where ACID coordination of multiple resource managers is not needed. In such scenarios running business logic under a **Transaction** policy of Not supported performs better than if it had been run under a Required policy. This benefit is exploited through the **Local Transactions - Resolution-control** extended deployment setting of ContainerAtBoundary. With this setting, application interactions with resource providers (such as databases) are managed within implicit RMLTs that are both started and ended by the container. The RMLTs are committed by the container at the configured **Local Transactions - Boundary**; for example at the end of a method. If the application returns control to the container by an exception, the container rolls back any RMLTs that it has started.

This usage applies to both servlets and enterprise beans.

Use local transactions in a managed environment that guarantees clean-up.

Applications that want to control RMLTs, by starting and ending them explicitly, can use the default **Local Transactions - Resolution-control** extended deployment setting of Application. In this case, the container ensures connection cleanup at the boundary of the local transaction context.

J2EE specifications that describe application use of local transactions do so in the manner provided by the default setting of **Local Transactions -**

Resolution-control=Application and **Local Transactions - Unresolved-action=Rollback**. By configuring the **Local Transactions - Unresolved-action** extended deployment setting to Commit, then any RMLTs started by the application but not completed when the local transaction containment ends (for example, when the method ends) are committed by the container. This usage applies to both servlets and enterprise beans.

Extend the duration of a local transaction beyond the duration of an EJB component method.

The J2EE specifications restrict the use of RMLTs to single EJB methods. This restriction is because the specifications have no scoping device, beyond a container-imposed method boundary, to which an RMLT can be extended. In WebSphere Application Server Enterprise, you can exploit the **Local Transactions - Boundary** extended deployment setting to give the following advantages:

- Significantly extend the use-cases of RMLTs
- Make conversational interactions with one-phase resource managers possible through ActivitySession support.

An ActivitySession is a WebSphere Application Server Enterprise programming model extension that provides a distributed context with a boundary that is longer than a single method. You can extend the use of RMLTs over the longer ActivitySession boundary, which can be controlled by a client. The ActivitySession boundary reduces the need to use distributed transactions where ACID operations on multiple resources are not needed. This benefit is exploited through the **Local Transactions - Boundary** extended deployment setting of ActivitySession. Such extended RMLTs can remain under the control of the application or be managed by the container depending on the use of the **Local Transactions - Resolution-control** deployment descriptor setting.

Coordinate multiple one-phase resource managers.

For resource managers that do not support XA transaction coordination, a client can exploit ActivitySession-bounded local transaction contexts. Such contexts give a client the same ability to control the completion direction of the resource updates by the resource managers as the client has for transactional resource managers. A client can start an ActivitySession and call its entity beans under that context. Those beans can perform their RMLTs within the scope of that ActivitySession and return without completing the RMLTs. The client can later complete the ActivitySession in a commit or rollback direction and cause the container to drive the ActivitySession-bounded RMLTs in that coordinated direction.

To determine how best to configure the transaction support for an application, depending on what you want to do with transactions, consider the following points.

General points

- You want to start and end global transactions explicitly in the application (BMT session beans and servlets only).

For a session bean, set the **Transaction type** to Bean (to use bean-managed transactions) in the component's deployment descriptor. (You do not need to do this for servlets.)

- You want to access only one XA or non-XA resource in a method.

In the component's deployment descriptor, set **Local Transactions - Resolution-control** to ContainerAtBoundary. In the Container transaction deployment descriptor, set **Transaction** to Supports.

- You want to access several XA resources atomically across one or more bean methods.

In the Container transaction deployment descriptor, set **Transaction** to Required, Requires new, or Mandatory.

- You want to access several non-XA resource in a method without having to worry about managing your own local transactions.

In the component's deployment descriptor, set **Local Transactions - Resolution-control** to ContainerAtBoundary. In the Container transaction deployment descriptor, set **Transaction** to Not supported.

- You want to access several non-XA resources in a method and want to manage them independently.

In the component's deployment descriptor, set **Local Transactions - Resolution-control** to Application and set **Local Transactions - Unresolved-action** to Rollback. In the Container transaction deployment descriptor, set **Transaction** to Not supported.

Points specific to WebSphere Application Server Enterprise

- You want to access one of more non-XA resources across multiple EJB method calls without having to worry about managing your own local transactions.

In the component's deployment descriptor, set **Local Transactions - Resolution-control** to ContainerAtBoundary, **Local Transactions - Boundary** to ActivitySession, and **Bean Cache - Activate at** to ActivitySession. In the Container transaction deployment descriptor, set **Transaction** to Not supported and set **ActivitySession** attribute to Required, Requires new, or Mandatory.

- You want to access several non-XA resources across multiple EJB method calls and want to manage them independently.

In the component's deployment descriptor, set **Local Transactions - Resolution-control** to Application, **Local Transactions - Boundary** to ActivitySession, and **Bean Cache - Activate at** to ActivitySession. In the Container Transaction deployment descriptor, set **Transaction** to Not supported and set **ActivitySession** attribute to Required, Requires new, or Mandatory.

- You want to use a single non-XA resource and one or more XAResources.

Use the Last Participant Support of WebSphere Application Server Enterprise.

Points specific to WebSphere Application Server for z/OS

- You want to use a non-XA resource along with multiple two-phase RRS resources.

A non-XA resource in a transaction along with RRS resources is supported any time a global transaction is active. A global transaction is active when the deployment descriptor has **Transaction** set to Supports, Required, Requires New, or Mandatory. Global transactions also are active for Bean-Managed deployments.

Managing active transactions

Use this task to manage transactions that are active on an application server.

You can use this task to display a snapshot of all the transactions currently running on an application server. For each transaction, the following properties are shown: its local ID, global ID, and current status. The transaction status is shown as an integer value. The values correspond to the following status:

- 0 - active
- 1 - marked for rollback
- 2 - prepared
- 3 - committed
- 4 - rolled back
- 5 - unknown
- 6 - none
- 7 - preparing
- 8 - committing
- 9 - rolling back

You can also choose to finish transactions manually.

Under normal circumstances, transactions should run and complete (commit or rollback) automatically, without the need for intervention. However, in some circumstances, you may need to finish a transaction manually. For example, you may want to finish a transaction that has become stuck polling a resource manager that you know will not become available again within the desired timeframe.

Note: If you choose to finish a transaction on an application server, it is recorded as having completed in the transaction service logs for that server, so will not be eligible for recovery during server start up. If you finish a transaction, you are responsible for cleaning up any in-doubt transactions on the resource managers affected.

To manage the active transactions for an application server, use the administrative console to complete the following steps:

1. In the navigation pane, select **Servers-> Manage Application Servers** This displays a list of application servers in the content pane.
2. In the content pane, click *your_app_server* This displays the properties of the application server, *your_app_server*.
3. In the content pane, click the **Runtime** tab. This displays the runtime properties of the application server.
4. In the Additional Properties table, select **Transaction Service** This displays the runtime properties of the Transaction Service.
5. Click **Manage Transactions**. This displays a snapshot of all the transactions currently running on the server. For each transaction, the following properties are shown: its local ID, current status, and global ID.
6. If you want to finish one or more transactions, select the checkbox provided on the entry for the transaction, then click **Finish**. Alternatively, to finish all transactions, select the checkbox in the header of the transactions table, then click **Finish**.

Interoperating transactionally between application servers

This topic describes some considerations and actions that you can take to interoperate transactionally between different types of application servers.

To interoperate transactionally with a non-WebSphere application server, WebSphere Application Server switches dynamically between WebSphere-optimized native transaction contexts and interoperable OTS contexts depending on the capability of the partner with which it is interoperating.

To interoperate transactionally with a non-WebSphere application server, WebSphere Application Server switches dynamically between native transaction contexts and interoperable OTS contexts depending on the capability of the partner with which it is interoperating. WebSphere for z/OS always interoperates with other OTS contexts.

Troubleshooting transactions

Use this overview task to help resolve a problem that you think is related to the Transaction service.

To identify and resolve transaction-related problems, you can use the standard WebSphere Application Server RAS facilities. If you encounter a problem that you think might be related to transactions, complete the following steps:

1. Check for transaction messages in the administrative console. The Transaction service produces diagnostic messages prefixed by “WTRN”. The error message indicates the nature of the problem and provides some detail. The associated message information provides an explanation and any user actions to resolve the problem.
2. Check for Transaction messages in the activity log. Activity log messages produced by the Transaction service are accompanied by log analyzer descriptions.
3. Check for more messages in the application server’s stdout.log. For more information about a problem, check the stdout.log file for the application server, which should contain more error messages and extra details about the problem.
4. Check for more messages in other potential message output repositories. For more information about a problem, check the standard output file configured by your administrator. This will contain more error messages and other detailed information about the problem.
5. Check for messages related to the application server’s transaction log directory when the problem occurred.

Note: If you changed the transaction log directory and a problem caused the application server to fail (with in-flight transactions) before the server was restarted properly, the server will next start with the new log directory and be unable to automatically resolve in-flight transactions that were recorded in the old log directory. To resolve this, you can copy the transaction logs to the new directory then stop and restart the application server.

6. Check the RRS logs for any transaction activity involving RRS-compliant resources. IBM WebSphere Application Server for z/OS is capable of supporting both XA and RRS resource managers, and of coordinating a mix of RRSTransactional resource managers and XA capable resource managers under the same global transaction. If your installation uses XA resource managers, RRS resource managers, or a mixture of both, you can use the administrative console to view transaction logs that contain information about all transactions. You can find additional information about RRS transactions by using the RRS panels. For additional information, see:
 - *WebSphere Application Server for z/OS V5.0: Diagnosis*, GA22-7914, for information about diagnosing problems related to transactions.
 - *WebSphere Application Server for z/OS V5.0: Operations and Administration*, GA22-7912, for information about using the RRS panels and transaction logs.

- *WebSphere Application Server for z/OS V5.0: Installation and Customization*, GA22-7910, for information about working with RRS logs and XA partner logs during restart and recovery mode.

Transaction service exceptions

This topic lists the exceptions that can be thrown by the WebSphere Application Server transaction service. The exceptions are listed in the following groups:

- Standard exceptions
- Heuristic exceptions

If the EJB container catches a system exception from the business method of an enterprise bean, and the method is running within a container-managed transaction, the container rolls back the transaction before passing the exception on to the client. For more information about how the container handles the exceptions thrown by the business methods for beans with container-managed transaction demarcation, see the section *Exception handling* in the Enterprise JavaBeans 2.0 specification. That section specifies the container's action as a function of the condition under which the business method executes and the exception thrown by the business method. It also illustrates the exception that the client receives and how the client can recover from the exception.

Standard exceptions

The standard exceptions such as `TransactionRequiredException`, `TransactionRolledbackException`, and `InvalidTransactionException` are defined in the Java Transaction API (JTA) 1.0.1 Specification.

InvalidTransactionException

This exception indicates that the request carried an invalid transaction context.

TransactionRequiredException exception

This exception indicates that a request carried a null transaction context, but the target object requires an active transaction.

TransactionRolledbackException exception

This exception indicates that the transaction associated with processing of the request has been rolled back, or marked for roll back. Thus the requested operation either could not be performed or was not performed because further computation on behalf of the transaction would be fruitless.

Heuristic exceptions

A heuristic decision is a unilateral decision made by one or more participants in a transaction to commit or rollback updates without first obtaining the consensus outcome determined by the Transaction Service. Heuristic decisions are an issue only after the participant has been prepared and the second phase of commit processing is underway. Heuristic decisions are normally made only in unusual circumstances, such as repeated failures by the transaction manager to communicate with a resource manager during two-phase commit. If a heuristic decision is taken, there is a risk that the decision differs from the consensus outcome, resulting in a loss of data integrity.

The following list provides a summary of the heuristic exceptions. For more detail, see the Java Transaction API (JTA) 1.0.1 Specification.

HeuristicRollback exception

This exception is raised on the commit operation to report that a heuristic decision was made and that all relevant updates have been rolled back.

HeuristicMixed exception

This exception is raised on the commit operation to report that a heuristic decision was made and that some relevant updates have been committed and others have been rolled back.

UserTransaction interface - methods available

For details about the methods available with the UserTransaction interface, see the WebSphere Application Server application programming interface reference information (Javadoc) or the Java Transaction API (JTA) 1.0.1 Specification.

Chapter 13. Using naming

Naming is used by clients of WebSphere Application Server applications most commonly to obtain references to objects related to those applications, such as Enterprise JavaBeans (EJB) homes. The following steps outline the context of Naming in the overall application development and deployment process. Steps for this task follow:

1. Develop your application using either JNDI or CosNaming (CORBA) interfaces.

Use these interfaces to look up server application objects that are bound into the name space and obtain references to them. Most Java developers use the JNDI interface. However, the CORBA CosNaming interface is also available for performing Naming operations on WebSphere Application Server name servers or other CosNaming name servers.

2. Assemble your application using the .

Application assembly is a packaging and configuration step that is a prerequisite to application deployment. If the application you are assembling is a client to an application running in another process, you should qualify the `jniName` values in the deployment descriptors for the objects related to the other application. Otherwise, you may need to override the names with qualified names during application deployment. If the objects have fixed qualified names configured for them, you should use them so that the `jniName` values do not depend on the other application's location within the topology of the cell.

3. Deploy your application

Put your assembled application onto the application server. If the application you are assembling is a client to an application running in another server process, be sure to qualify the `jniName` values for the other application's server objects if they are not already qualified.

For more information on qualified names, see "Lookup names support in deployment descriptors and thin clients."

4. Configure name space bindings.

This step is necessary in these cases:

- Your deployed application is to be accessed by legacy client applications running on previous versions of WebSphere Application Server. In this case, you must configure additional name bindings for application objects relative to the default initial context for legacy clients. (Version 5 clients have a different initial context from legacy clients.)
- The application requires qualified name bindings for such reasons as:
 - It will be accessed by J2EE client applications or server applications running in another server process.
 - It will be accessed by thin client applications.

In this case, you can configure name bindings as additional bindings for application objects. The qualified names for the configured bindings are *fixed*, meaning they do not contain elements of the cell topology that can change if the application is moved to another server. Objects as bound into the name space by the system can always be qualified with a topology-based name. You must explicitly configure a name binding to use as a fixed qualified name.

For more information on qualified names, see "Lookup names support in deployment descriptors and thin clients." For more information on configured name bindings, see "Configured name bindings."

5. Troubleshoot any problems that develop.

If a Naming operation is failing and you need to verify whether certain name bindings exist, use the `dumpNameSpace` tool to generate a dump of the name space.

Naming

Naming is used by clients of WebSphere Application Server applications to obtain references to objects related to those applications, such as Enterprise JavaBeans (EJB) homes.

These objects are bound into a mostly hierarchical structure, referred to as a *name space*. In this structure, all non-leaf objects are called *contexts*. Leaf objects can be contexts and other types of objects. Naming operations, such as lookups and binds, are performed on contexts. All naming operations begin with obtaining an *initial context*. You can view the initial context as a starting point in the name space.

The name space structure consists of a set of *name bindings*, each consisting of a name relative to a specific context and the object bound with that name. For example, the name `myApp/myEJB` consists of one non-leaf binding with the name `myApp`, which is a context. The name also includes one leaf binding with the name `myEJB`, relative to `myApp`. The object bound with the name `myEJB` in this example happens to be an EJB home reference. The whole name `myApp/myEJB` is relative to the initial context, which you can view as a starting place when performing naming operations.

You can access and manipulate the name space through a *name server*. Users of a name server are referred to as *naming clients*. Naming clients typically use the Java Naming and Directory Interface (JNDI) to perform naming operations. Naming clients can also use the Common Object Request Broker Architecture (CORBA) `CosNaming` interface.

Typically, objects bound to the name space are resources and objects associated with installed applications. These objects are bound by the system, and client applications perform lookup operations to obtain references to them. Occasionally, server and client applications bind objects to the name space. An application can bind objects to transient or persistent partitions, depending on requirements.

In J2EE environments, some JNDI operations are performed with `java: URL` names. Names bound under these names are bound to a completely different name space which is local to the calling process. However, some lookups on the `java: name` space may trigger indirect lookups to the name server.

New features for name space support

The following are new features of the WebSphere Application Server V5 naming implementation:

- **Name space is distributed.**

For additional scalability, the name space for a cell is distributed among various servers. Every server has a name server. In previous releases, there was only one name server for an entire administrative domain.

In WebSphere Application Server versions prior to V5, all servers shared the same default initial context, and everything was bound relative to that same initial context. In WebSphere Application Server V5, the default initial context for a server is its server root. System artifacts, such as EJB homes and resources, are bound to the server root of the server with which they are associated.

- **Transient and persistent partitions.**

The name space is partitioned into transient areas and persistent areas. Server roots are transient. System-bound artifacts such as EJB homes and resources are bound under server roots. There is a cell persistent root, which you can use for cell-scoped persistent bindings, and a node persistent root, which you can use to bind objects with a node scope.

- **System name space structure.**

The name space for the entire cell is federated among all servers in the cell. Every server process contains a name server. All name servers provide the same logical view of the cell name space. The various server roots and persistent partitions of the name space are interconnected by means of a system name space. You can use the system name space structure to traverse to any context in the cell 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 URLs.**

WebSphere Application Server V5 contains support for Common Object Request Broker Architecture (CORBA) object URLs (corbaloc and corbname) as Java Naming and Directory Interface (JNDI) provider URLs and lookup names.

Name space logical view

The name space for the entire cell is federated among all servers in the cell. Every server process contains a name server. All name servers provide the same logical view of the cell name space. The various server roots and persistent partitions of the name space are interconnected by a system name space. You can use the system name space structure to traverse to any context in a the cell's name space. A logical view of the name space is shown in the following diagram.

Logical View of a Cell's Name Space

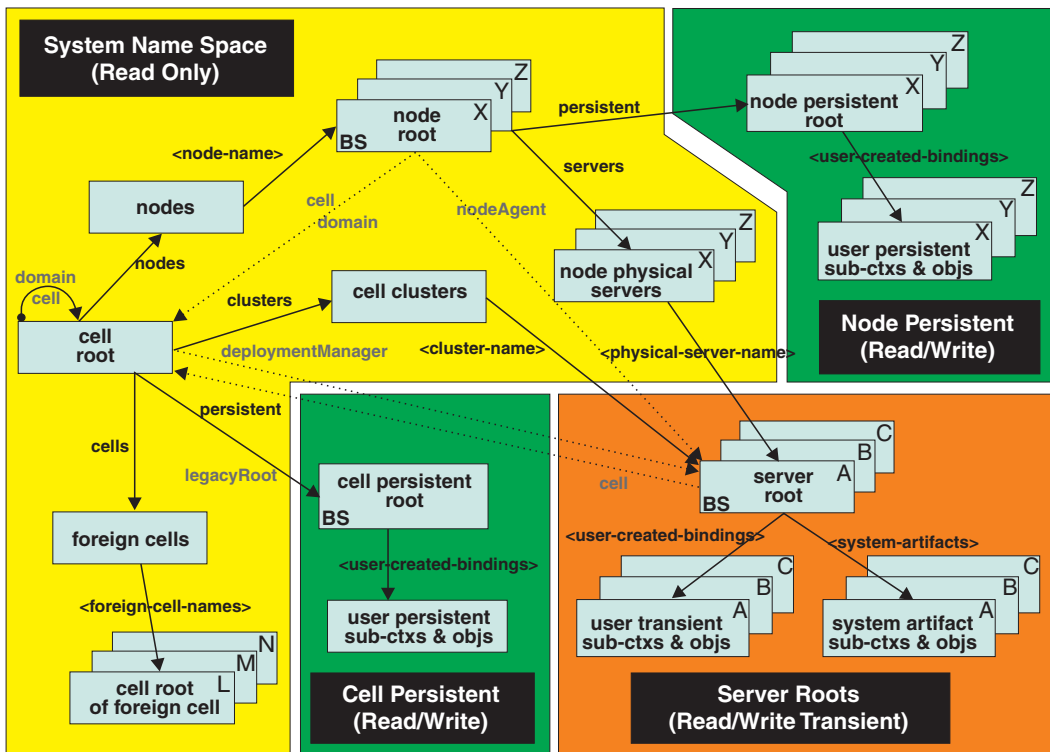


Figure 4. Name Space Logical View

The bindings in the preceding diagram appear with solid arrows, labeled in bold, and dashed arrows, labeled in gray. Solid arrows represent *primary bindings*. A primary binding is formed when the associated subcontext is created. Dashed arrows show *linked bindings*. A linked binding is formed when an existing context is bound under an additional name. Linked bindings are added for convenience or interoperability with previous WebSphere Application Server versions.

A cell name space is composed of contexts which reside in servers throughout the cell. All name servers in the cell provide the same logical view of the cell name space. A name server constructs this view at startup by reading configuration information. Each name server has its own local in-memory copy of the name space and does not require another running server to function. There are, however, a few exceptions. Server roots for other servers are not replicated among all the servers. The respective server for a server root must be running to access that server root context.

In WebSphere Application Server Network Deployment cells, the cell and node persistent areas can be read even if the deployment manager and respective node agent are not running. However, the deployment manager must be running to update the cell persistent segment, and a node agent must be running to update its respective node persistent segment.

Name space partitions

There are four major partitions in a cell name space:

- System name space partition
- Server roots partition
- Cell persistent partition

- Node persistent partition

System name space partition

The system name space contains a structure of contexts based on the cell topology. The system structure supports traversal to all parts of a cell name space and to the cell root of other cells, which are configured as foreign cells. The root of this structure is the cell root. In addition to the cell root, the system structure contains a node root for each node in the cell. You can access other contexts of interest specific to a node from the node root, such as the node persistent root and server roots for servers configured in that node.

All contexts in the system name space are read-only. You cannot add, update, or remove any bindings.

Server roots partition

Each server in a cell has a server root context. A server root is specific to a particular server. You can view the server roots for all servers in a cell as being in a transient read/write partition of the cell name space. System artifacts, such as EJB homes for server applications and resources, are bound under the server root context of the associated server. A server application can also add bindings under its server root. These bindings are transient. Therefore, the server application creates all required bindings at application startup, so they exist anytime the application is running.

A server cluster is composed of many servers that are logically equivalent. Each member of the cluster has its own server root. These server roots are not replicated across the cluster. In other words, adding a binding to the server root of one member does not propagate it to the server roots of the other cluster members. To maintain the same view across the cluster, you should create all user bindings under the server root by the server application at application startup so that the bindings are present under the server root of each cluster member. Because of Workload Management (WLM) behavior, a JNDI client outside a cluster has no control over which cluster member's server root context becomes the target of the JNDI operation. Therefore, you should execute bind operations to the server root of a cluster member should from within that cluster member process only.

Distributing application objects among many server roots is a departure from previous WebSphere Application Server releases, where all system artifacts were bound under a single root. This change can affect the names that clients use to look up these objects.

Server-scoped bindings are relative to a server's server root.

Cell persistent partition

The root context of the cell persistent partition is the cell persistent root. A binding created under the cell persistent root is saved as part of the cell configuration and continues to exist until it is explicitly removed. Applications that need to create additional persistent bindings of objects generally associated with the cell can bind these objects under the cell persistent root.

It is important to note that the cell persistent area is not designed for transient, rapidly changing bindings. The bindings are more static in nature, such as part of an application setup or configuration, and are not created at run time.

Note: In WebSphere Application Server Network Deployment cells, to bind objects to the cell persistent root, ensure that the deployment manager and all node agents in the cell are running.

An important role of the cell persistent root is as the initial context for clients running in previous WebSphere Application Server versions. If you want to access an enterprise bean by WebSphere Application Server v4.0.x and 3.5.x clients, you must ensure that a binding for it has been added to the cell persistent root. You can configure these additional bindings as cell-scoped bindings.

Node persistent partition

The node persistent partition is similar to the cell partition except that each node has its own node persistent root. A binding created under a node persistent root is saved as part of that node configuration and continues to exist until it is explicitly removed.

Applications that need to create additional persistent bindings of objects associated with a specific node can bind those objects under that particular node's node persistent root. As with the cell persistent area, it is important to note that the node persistent area is not designed for transient, rapidly changing bindings. These bindings are more static in nature, such as part of an application setup or configuration, and are not created at run time.

Note: In WebSphere Application Server Network Deployment cells, to bind objects to a node persistent root, ensure the node agent for the node is running.

Unlike the cell persistent root, the node persistent root plays no special role in interoperability with WebSphere Application Server clients of previous releases. Node-scoped bindings are relative to a node's node persistent root.

Note: In the system name space, there is no persistent node root for the deployment manager node because no node agent or application servers run in that node.

Initial context support

All naming operations begin with obtaining an initial context. You can view the initial context as a starting point in the name space. Use the initial context to perform naming operations, such as looking up and binding objects in the name space.

Initial contexts registered with the ORB as initial references

The server root, cell persistent root, cell root, and node root are registered with the name server's ORB and can be used as an initial context. An initial context is used by CORBA and enterprise bean applications as a starting point for name space lookups. The keys for these roots as recognized by the ORB are shown in the following table:

Root Context	Initial Reference Key
Server Root	NameServiceServerRoot
Cell Persistent Root	NameServiceCellPersistentRoot
Cell Root	NameServiceCellRoot, NameService

Node Root	NameServiceNodeRoot
------------------	---------------------

A server root initial context is the server root context for the specific server you are accessing. Similarly, a node root initial context is the node root for the server being accessed.

You can use the previously mentioned keys in CORBA INS object URLs (corbaloc and corbaname) and as an argument to an ORB resolve_initial_references call. For examples, see CORBA and JNDI programming examples, which show how to get an initial context.

Default initial contexts

The default initial context depends on the type of client. Different categories of clients and the corresponding default initial context follow.

- **WebSphere Application Server V5 JNDI interface implementation**

The JNDI interface is used by EJB applications to perform name space lookups. WebSphere Application Server clients by default use the WebSphere Application Server CosNaming JNDI plug-in implementation. The default initial context for clients of this type is the server root of the server specified by the provider URL. For more details, refer to the JNDI programming examples on getting initial contexts.

- **WebSphere Application Server JNDI interface implementation prior to V5**

WebSphere Application Server clients running in releases prior to WebSphere Application Server V5 by default use WebSphere Application Server's v4.0 CosNaming JNDI plug-in implementation. The default initial context for clients of this type is the cell persistent root, also known as the *legacy root*.

- **Other JNDI implementation**

Some applications can perform name space lookups with a non-WebSphere Application Server CosNaming JNDI plug-in implementation. Assuming the key **NamingContext** is used to obtain the initial context, the default initial context for clients of this type is the cell root.

- **CORBA**

The standard CORBA client obtains an initial org.omg.CosNaming.NamingContext reference with the key **NamingContext**. The initial context in this case is the cell root.

Lookup names support in deployment descriptors and thin clients

Server objects, such as EJB homes, are bound relative to the server root context for the server in which the application is installed. Other objects, such as resources, can also be bound to a specific server root. The names used to look up these objects must be qualified so as to select the correct server root. This is a departure from previous versions of WebSphere Application Server, where these objects were all bound under a single root context. This section discusses what relative and qualified names are, when they can be used, and how you can construct them.

Relative names

All names are relative to a context. Therefore, a name that can be resolved from one context in the name space cannot necessarily be resolved from another context in the name space. This point is significant because the system binds objects with names relative to the server root context of the server in which the application is

installed. Each server has its own server root context. The initial JNDI context is by default the server root context for the server identified by the provider URL used to obtain the initial context. (Typically, the URL consists of a host and port.) For applications running in a server process, the default initial JNDI context is the server root for that server. A relative name will resolve successfully when the initial context is obtained from the server which contains the target object, but it will not resolve successfully from an initial context obtained from another server.

If all clients of a server application run in the same server process as the application, all objects associated with that application are bound to the same initial context as the clients' initial context. In this case, only names relative to the server's server root context are required to access these server objects. Frequently, however, a server application has clients that run outside the application's server process. The initial context for these clients can be different from the server application's initial context, and lookups on the relative names for server objects may fail. These clients need to use the qualified name for the server objects. This point must be considered when setting up the `jndiName` values in a J2EE client application deployment descriptors and when constructing lookup names in thin clients. Qualified names resolve successfully from any initial context in the cell.

Qualified names

All names are relative to a context. Here, the term qualified name refers to names that can be resolved from any initial context in a cell. This action is accomplished by using names that navigate to the same context, the cell root. The rest of the qualified name is then relative to the cell root and uniquely identifies an object throughout the cell. All initial contexts in a server (that is, all naming contexts in a server registered with the ORB as an initial reference) contain a binding with the name `cell`, which links back to the cell root context. All qualified names begin with the string `cell/` to navigate from the current initial context back to the cell root context.

A qualified name for an object is the same throughout the cell. The name can be topology-based, or some fixed name bound under the cell persistent root. Topology-based names, described in more detail below, navigate through the system name space to reach the target object. A fixed name bound under the cell persistent root has the same qualified name throughout the cell and is independent of the topology. Creating a fixed name under the cell persistent root for a server application object requires an extra step when the server application is installed, but this step eliminates impacts to clients when the application is moved to a different location in the cell topology. The process for creating a fixed name is described later in this section.

Generally speaking, you **must** use qualified names for EJB `jndiName` values in a J2EE client application deployment descriptors and for EJB lookup names in thin clients. The only exception is when the initial context is obtained from the server in which the target object resides. For example, a session bean which is a client to an entity bean can use a relative name if the two beans run in the same server. If the session bean and entity beans run in different servers, the `jndiName` for the entity bean must be qualified in the session bean's deployment descriptors. The same requirement may be true for resources as well, depending on the scope of the resource.

- **Topology-based names**

The system name space partition in a cell's name space reflects the cell's topology. This structure can be navigated to reach any object bound into the

cell's name space. Topology-based qualified names include elements from the topology which reflect the object's location within the cell. For a system-bound object, such as an EJB home, the form for a topology-based qualified name depends on whether the object is bound to a single server or cluster. Both forms are described below.

Single Server

An object bound in a single server has a topology-based qualified name of the following form:

```
cell/nodes/nodeName/servers/serverName/relativeJndiName
```

where *nodeName* and *serverName* are the node name and server name for the server where the object is bound, and *relativeJndiName* is the unqualified name of the object; that is, the object's name relative to its server's server root context.

Server Cluster

An object bound in a server cluster has a topology-based qualified name of the following form:

```
cell/clusters/clusterName/relativeJndiName
```

where *clusterName* is the name of the server cluster where the object is bound, and *relativeJndiName* is the unqualified name of the object; that is, the object's name relative to a cluster member's server root context.

- **Fixed names**

It is possible to create a fixed name for a server object so that the qualified name is independent of the cell topology. This quality is desirable when clients of the application run in other server processes or as pure clients. Fixed names have the advantage of not changing if the object is moved to another server. The *jndiName* values in deployment descriptors for a J2EE client application can reference the qualified fixed name for a server object regardless of the cell topology on which the client or server application is being installed.

Defining a cell-wide fixed name for a server application object requires an extra step after the server application is installed. That is, a binding for the object must be created under the cell persistent root. A fixed name bound under the cell persistent root can be any name, but all names under the cell persistent root must be unique within the cell because the cell persistent root is global to the entire cell.

A qualified fixed name has the form:

```
cell/persistent/fixedName
```

where *fixedName* is an arbitrary fixed name.

The binding can be created programmatically (for example, using JNDI). However, it is probably more convenient to configure a cell-scoped binding for the server object.

You must keep the programmatic or configured binding up-to-date. Configured EJB bindings are based on the location of the enterprise bean within the cell topology, and moving the EJB application to another single server or to a server cluster, for example, requires the configured binding to be updated. Similar changes affect an EJB home reference programmatically bound so that the fixed name would need to be rebound with a current reference. However, for J2EE clients, the *jndiName* value for the object, and for thin clients, the lookup name

for the object, remains the same. In other words, clients that access objects by fixed names are not affected by changes to the configuration of server applications they access.

JNDI support in WebSphere Application Server

IBM WebSphere Application Server includes a name server to provide shared access to Java components, and an implementation of the `javax.naming` JNDI package which supports user access to the WebSphere Application Server name server through the JNDI naming interface.

WebSphere Application Server does *not* provide implementations for:

- `javax.naming.directory` or
- `javax.naming.ldap` packages

Also, WebSphere Application Server does *not* support interfaces defined in the `javax.naming.event` package.

However, to provide access to LDAP servers, the development kit shipped with WebSphere Application Server supports Sun's implementation of:

- `javax.naming.ldap` and
- `com.sun.jndi.ldap.LdapCtxFactory`

WebSphere Application Server's JNDI implementation is based on version 1.2 of the JNDI interface, and was tested with Version 1.2.1 of Sun's JNDI Service Provider Interface (SPI).

The default behavior of this JNDI implementation is adequate for most users. However, users with specific requirements can control certain aspects of JNDI behavior.

Developing applications that use JNDI

References to EJB homes and other artifacts such as data sources are bound to the WebSphere name space. These objects can be obtained through the JNDI interface. Before you can perform any JNDI operations, you need to get an initial context. You can use the initial context to look up objects bound to the WebSphere name space.

These examples describe how to get an initial context and how to perform lookup operations.

- Getting the default initial context
- Getting an initial context by setting the provider URL property
- Setting the provider URL property to select a different root context as the initial context
- Looking up an EJB home with JNDI
- Looking up a JavaMail session with JNDI

In these examples, the default behavior of features specific to WebSphere's JNDI Context implementation is used.

WebSphere Application Server's JNDI context implementation includes special features. JNDI caching enhances performance of repeated lookup operations on the same objects. Name syntax options offer a choice of a name syntaxes, one optimized for typical JNDI clients, and one optimized for interoperability with

CosNaming applications. Most of the time, the default behavior of these features is the preferred behavior. However, sometimes you should modify the behavior for specific situations.

JNDI caching and name syntax options are associated with a `javax.naming.InitialContext` instance. To select options for these features, set properties that are recognized by the WebSphere Application Server's initial context factory. To set JNDI caching or name syntax properties which will be visible to WebSphere Application Server's initial context factory, follow the following steps.

1. Configure JNDI caches

JNDI caching can greatly increase performance of JNDI lookup operations. By default, JNDI caching is enabled. In most situations, this default is the desired behavior. However, in specific situations, use the other JNDI cache options.

Objects are cached locally as they are looked up. Subsequent lookups on cached objects are resolved locally. However, cache contents can become stale. This situation is not usually a problem, since most objects you look up do not change frequently. If you need to look up objects which change relatively frequently, change your JNDI cache options.

JNDI clients can use several properties to control cache behavior.

You can set properties:

- From the command line by entering the actual string value. For example:
`java -Dcom.ibm.websphere.naming.jndicache.maxentrylife=1440`
- In a `jndi.properties` file by creating a file named `jndi.properties` as a text file with the desired properties settings. For example:

```
...
com.ibm.websphere.naming.jndicache.cacheobject=none
...
```

Include the file as the beginning of the classpath, so that the class loader loads your copy of `jndi.properties` before any other copies.

- Within a Java program by using the **PROPS.JNDI_CACHE*** Java constants, defined in the `com.ibm.websphere.naming.PROPS` file. The constant definitions follow:

```
public static final String JNDI_CACHE_OBJECT =
    "com.ibm.websphere.naming.jndicache.cacheobject";
public static final String JNDI_CACHE_OBJECT_NONE      = "none";
public static final String JNDI_CACHE_OBJECT_POPULATED = "populated";
public static final String JNDI_CACHE_OBJECT_CLEARED   = "cleared";
public static final String JNDI_CACHE_OBJECT_DEFAULT  =
    JNDI_CACHE_OBJECT_POPULATED;

public static final String JNDI_CACHE_NAME =
    "com.ibm.websphere.naming.jndicache.cachename";
public static final String JNDI_CACHE_NAME_DEFAULT = "providerURL";

public static final String JNDI_CACHE_MAX_LIFE =
    "com.ibm.websphere.naming.jndicache.maxcachelife";
public static final int    JNDI_CACHE_MAX_LIFE_DEFAULT = 0;

public static final String JNDI_CACHE_MAX_ENTRY_LIFE =
    "com.ibm.websphere.naming.jndicache.maxentrylife";
public static final int    JNDI_CACHE_MAX_ENTRY_LIFE_DEFAULT = 0;
```

To use the previous properties in a Java program, add the property setting to a hashtable and pass it to the `InitialContext` constructor as follows:

```

java.util.Hashtable env = new java.util.Hashtable();
...
env.put(PROPS.JNDI_CACHE_OBJECT, PROPS.JNDI_CACHE_OBJECT_NONE); // Disable caching
...
javax.naming.Context initialContext = new javax.naming.InitialContext(env);

```

2. Specify the name syntax

Most WebSphere applications use JNDI to look up EJB objects and do not need to look up objects bound by CORBA applications. Therefore, the default name syntax used for JNDI names is the most convenient. If your application needs to look up objects bound by CORBA applications, you may need to change your name syntax so that all CORBA CosNaming names can be represented.

JNDI clients can set the name syntax by setting a property. The property setting is applied by the initial context factory when you instantiate a new `java.naming.InitialContext` object. Names specified in JNDI operations on the initial context are parsed according to the specified name syntax.

You can set the property:

- From the command line by entering the actual string value. For example:

```
java -Dcom.ibm.websphere.naming.name.syntax=ins
```

- In a `jndi.properties` file by creating a file named `jndi.properties` as a text file with the desired properties settings. For example:

```

...
com.ibm.websphere.naming.name.syntax=ins
...

```

Include the file as the beginning of the classpath, so that the class loader loads your copy of `jndi.properties` before any other copies.

- Within a Java program by using the `PROPS.NAME_SYNTAX*` Java constants, defined in the `com.ibm.websphere.naming.PROPS` file. The constant definitions follow:

```

public static final String NAME_SYNTAX = "com.ibm.websphere.naming.name.syntax";
public static final String NAME_SYNTAX_JNDI = "jndi";
public static final String NAME_SYNTAX_INS = "ins";

```

To use the previous properties in a Java program, add the property setting to a hashtable and pass it to the `InitialContext` constructor as follows:

```

java.util.Hashtable env = new java.util.Hashtable();
...
env.put(PROPS.NAME_SYNTAX, PROPS.NAME_SYNTAX_INS); // Set name syntax to INS
...
javax.naming.Context initialContext = new javax.naming.InitialContext(env);

```

Example: Getting the default initial context

This example below gets the default initial context. That is, no provider URL is passed to the `javax.naming.InitialContext` constructor. The following section explains the process of determining the address of the bootstrap server to use to obtain the initial context.

```

...
import javax.naming.Context;
import javax.naming.InitialContext;
...
Context initialContext = new InitialContext();
...

```

The default initial context returned depends the runtime environment of the JNDI client. The initial context returned in the various environments are listed below:

- Thin client: The server root context of the server running on the local host at port 2809.

- Pure client:
 - The context specified by the `java.naming.provider.url` property passed to `launchClient` command with the `-CCD` command line parameter. The context usually will be the server root context of the server at the address specified in the URL, although it is possible to construct a `corbaname` or `corbaloc` URL which resolves to some other context.
 - If no provider URL was specified, the server root context of the server running on the host and port specified by the `-CCBootstrapHost` `-CCBootstrapPort` command line parameters. The default host is the local host, and the default port is 2809.
- Server process: The server root context for that process.

Even though no provider URL is explicitly specified in the above example, the `InitialContext` may find a provider URL defined in other places that it searches for property settings.

Users of properties which affect ORB initialization should read the rest of this section for a deeper understanding of exactly how initial contexts are obtained, which has changed from previous releases.

Determining which server is used to obtain the initial context

WebSphere Application Server name servers are CORBA CosNaming name servers, and WebSphere Application Server provides a CosNaming JNDI plug-in implementation for JNDI clients to perform naming operations on WebSphere Application Server name spaces. The WebSphere Application Server CosNaming plug-in implementation is selected through a JNDI property that is passed to the `InitialContext` constructor. This property is `java.naming.factory.initial`, and it specifies the initial context factory implementation to use to obtain an initial context. The factory returns a `javax.naming.Context` instance, which is part of its implementation.

The WebSphere Application Server initial context factory, `com.ibm.websphere.naming.WsnInitialContextFactory`, is typically used by WebSphere Application Server applications to perform JNDI operations. The WebSphere Application Server run-time environment is set up to use this WebSphere Application Server initial context factory if one is not specified explicitly by the JNDI client. When the initial context factory is invoked, an *initial context* is obtained. The following paragraphs explain how the WebSphere Application Server initial context factory obtains the initial context in client and server environments.

- **Understanding the registration of initial references in server processes**

Every WebSphere Application Server has an ORB used to receive and dispatch invocations on objects running in that server. Services running in the server process can register initial references with the ORB. Each initial reference is registered under a key, which is a string value. An initial reference can be any CORBA object. WebSphere Application Server name servers register several initial contexts as initial references under predefined keys. Each name server initial reference is an instance of the interface `org.omg.CosNaming.NamingContext`.

- **Obtaining initial references in pure client processes**

Pure JNDI clients, that is, JNDI clients which are not running in a WebSphere Application Server process, also have an ORB instance. This client ORB instance can be passed to the `InitialContext` constructor, but typically the initial context factory creates and initializes the client ORB instance transparently. A client ORB

can be initialized with initial references, but the initial references most likely resolve to objects running in some server. The initial context factory does not define any default initial references when it initializes an ORB. If the `resolve_initial_references` method is invoked on the client ORB when no initial references have been configured, the method invocation fails. This condition is typical for pure client processes. To obtain an initial NamingContext reference, the initial context factory must invoke `string_to_object` with an IIOP type CORBA object URL, such as `corbaloc:iiop:myhost:2809`. The URL specifies the address of the server from which to obtain the initial context. The host and port information is extracted from the provider URL passed to the InitialContext constructor. If no provider URL is defined, the WebSphere Application Server initial context factory uses the default provider URL of `corbaloc:iiop:localhost:2809`. The `string_to_object` ORB method resolves the URL and communicates with the target server ORB to obtain the initial reference.

- **Obtaining initial references in server processes**

If the JNDI client is running in a WebSphere Application Server process, the initial context factory obtains a reference to the server ORB instance if the JNDI client does not provide an ORB instance. Typically, JNDI clients running in server processes use the server ORB instance; that is, they do not pass an ORB instance to the InitialContext constructor. The name server which is running in the server process sets a provider URL as a `java.lang.System` property to serve as the default provider URL for all JNDI clients in the process. This default provider URL is `corbaloc:rir:/NameServiceServerRoot`. This URL resolves to the server root context for that server. (The URL is equivalent to invoking `resolve_initial_references` on the ORB with a key of `NameServiceServerRoot`. The name server registers the server root context as an initial reference under that key.)

- **Understanding the legacy ORB protocol**

Previous versions of WebSphere Application Server used a different ORB implementation, which used a legacy protocol in contrast with the Interoperable Name Service (INS) protocol now used. This change has affected the implementation of the WebSphere Application Server initial context factory.

Certain types of pure clients can experience different behavior when getting initial JNDI contexts as compared to previous releases of WebSphere Application Server. This behavior is discussed in more detail below.

The following ORB properties are used with the legacy ORB protocol for ORB initialization and are now deprecated:

- `com.ibm.CORBA.BootstrapHost`
- `com.ibm.CORBA.BootstrapPort`

The new INS ORB is different in a major respect, in that it exhibits no default behavior if no initial references are defined. In the legacy ORB, the bootstrap host and port values defaulted to `localhost` and `900`. All initial references were obtained from the server running on the bootstrap host and port. So, if the ORB user provided no bootstrap host and port, all initial references are resolved from the server running on the local host at port `900`. The INS ORB has no concept of bootstrap host or bootstrap port. All initial references are defined independently. That is, different initial references could resolve to different servers. If `ORB.resolve_initial_references` is invoked with a key such that the ORB is not initialized with an initial reference having that key, the call fails.

In previous releases of WebSphere Application Server, the initial context factory invoked `resolve_initial_references` on the ORB in the absence of any provider URL. This action succeeded if a name server at the default bootstrap host and

port was running. Today, with the INS ORB, this would fail. (Actually, the ORB would fall back to the legacy protocol during the deprecation period, but when the legacy protocol is no longer supported, the operation would fail.) The initial context factory now uses a default provider URL of `corbaloc:iiop:localhost:2809`, and invokes `string_to_object` with the provider URL. This operation preserves the behavior that pure clients in previous releases experienced when they set no ORB bootstrap properties or provider URL.

However, this different initial context factory implementation changes the behavior experienced by certain legacy pure clients, which do not specify a provider URL:

- Clients which set the ORB bootstrap properties listed above when getting an initial context.
- Clients which supply their own ORB instance to the `InitialContext` constructor.

There are two ways to circumvent this change of behavior:

- Always specify an IIOP type provider URL. This approach does not depend on the bootstrap host and port properties and continues to work when support for the bootstrap host and port properties is removed. For example, you can express bootstrap host and port property values of `myHost` and `2809`, respectively, as `corbaloc:iiop:myHost:2809`.
- Use an `rir` type provider URL:
 - Specify `corbaloc:rir:/NameServiceServerRoot` if the ORB is initialized to use a WebSphere Application Server 5 server as the bootstrap server.
 - Specify `corbaname:rir:/NameService#domain/legacyRoot` if the ORB is initialized to use a WebSphere Application Server 4.0.x server as the bootstrap server.
 - Specify `corbaloc:rir:/NameService` if the ORB is initialized to use a server other than a WebSphere Application Server 5 or 4.0.x server as the bootstrap server.

URLs of this type are equivalent to invoking `resolve_initial_references` on the ORB with the specified key. If the bootstrap host and port properties are being used to initialize the ORB, this approach will not work when the bootstrap and host properties are no longer supported.

- **The `InitialContext` constructor search order for JNDI properties**

If the code snippet shown at the beginning of this section is executed by an application, the bootstrap server depends on the value of the property, `java.naming.provider.url`. If the property is not set (in server processes the default value is set as a system property), the default host of `localhost` and default port of `2809` are used as the address of the server from which to obtain the initial context. The JNDI specification describes where the `InitialContext` constructor looks for `java.naming.provider.url` property settings, but briefly, the property is picked up from the following places in the order shown:

1. The `InitialContext` constructor. This does not apply to the above example since the example uses the empty `InitialContext` constructor.
2. System environment. You can add JNDI properties to the system environment as an option on the Java command invocation and by program code. The recommended way to set the provider URL in the system environment is as an option supplied to the Java command invocation. Setting the provider URL in this manner is not temporal, so that getting a default initial context will always yield the same result. It is generally recommended that program code not set the provider URL property in the system environment because as a side-effect, this could adversely affect other, possibly unrelated, code running elsewhere in the same process.

3. `jndi.properties` file. There may be many `jndi.properties` files that are within the scope of the class loader in effect. All `jndi.properties` files are used for setting JNDI properties, but the provider URL setting is determined by the first `jndi.properties` file returned by the class loader.

Example: Getting an initial context by setting the provider URL property

In general, JNDI clients should assume the correct environment is already configured so there is no need to explicitly set property values and pass them to the `InitialContext` constructor. However, a JNDI client may need to access a name space other than the one identified in its environment. In this case, it is necessary to explicitly set the `java.naming.provider.url` (provider URL) property used by the `InitialContext` constructor. A provider URL contains bootstrap server information that the initial context factory can use to obtain an initial context. Any property values passed in directly to the `InitialContext` constructor take precedence over settings of those same properties found elsewhere in the environment.

You can use two different provider URL forms with WebSphere Application Server's initial context factory:

- A CORBA object URL (new for J2EE 1.3)
- An IIOP URL

CORBA object URLs are more flexible than IIOP URLs and are the recommended URL format to use. CORBA object URLs are part of the OMG CosNaming Interoperable Naming Specification. A `corbaname` URL, for example, can include initial context and lookup name information and can be used as a lookup name without the need to explicitly obtain another initial context. The IIOP URLs are the legacy JNDI format, but are still supported by the WebSphere Application Server initial context factory.

The following examples illustrate the use of these URLs.

Using a CORBA object URL

This example shows a CORBA object URL.

```
...
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
...
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, "corbaloc:iiop:myhost.mycompany.com:2809");
Context initialContext = new InitialContext(env);
...
```

Using a CORBA object URL with multiple name server addresses

CORBA object URLs can contain more than one bootstrap address. You can use this feature when attempting to obtain an initial context from a server cluster. You can specify the bootstrap addresses for all servers in the cluster in the URL. The operation succeeds if at least one of the servers is running, eliminating a single point of failure. There is no guarantee of any particular order in which the address list will be processed. For example, the second bootstrap address may be used to obtain the initial context even though the server at the first bootstrap address in the list is available.

Multiple-address provider URLs should only contain the bootstrap addresses of members of the same cluster. Otherwise, incorrect behavior may occur.

An example of a corbaloc URL with multiple addresses follows.

```
...
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
...
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.ibm.websphere.naming.WsnInitialContextFactory");
// All of the servers in the provider URL below are members of
// the same cluster.
env.put(Context.PROVIDER_URL,
        "corbaloc:myhost1:9810,myhost1:9811,myhost2:9810");
Context initialContext = new InitialContext(env);
...
```

Using a CORBA object URL from a non-WebSphere Application Server JNDI implementation

Initial context factories for CosNaming JNDI plug-in implementations other than the WebSphere Application Server initial context factory most likely obtain an initial context using the object key, `NameService`. When you use such a context factory to obtain an initial context from a WebSphere Application Server name server, the initial context is the cell root context. Since system artifacts such as EJB homes associated with a server are bound under the server's server root context, names used in JNDI operations must be qualified. If you want to use relative names, ensure your initial context is the server root context under which the target object is bound. In order to make the server root context the initial context, specify a corbaloc provider URL with an object key of `NameServiceServerRoot`.

This example shows a CORBA object type URL from a non-WebSphere Application Server JNDI implementation. This example assumes full CORBA object URL support by the non-WebSphere Application Server JNDI implementation. The object key of `NameServiceServerRoot` is specified so that the initial context will be the specified server's server root context.

```
...
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
...
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.somecompany.naming.TheirInitialContextFactory");
env.put(Context.PROVIDER_URL,
        "corbaname:iiop:myhost.mycompany.com:9810/NameServiceServerRoot");
Context initialContext = new InitialContext(env);
...
```

If qualified names are used, you can use the default key of `NameService`.

Using an IIOP URL

The IIOP type of URL is a legacy format which is not as flexible as CORBA object URLs. However, URLs of this type are still supported. The following example shows an IIOP type URL as the provider URL.

```
...
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
...
```

```

Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, "iiop://myhost.mycompany.com:2809");
Context initialContext = new InitialContext(env);
...

```

Example: Setting the provider URL property to select a different root context as the initial context

Each server contains its own server root context, and, when bootstrapping to a server, the server root is the default initial JNDI context. Most of the time, this default is the desired initial context, since system artifacts such as EJB homes are bound there. However, other root contexts exist, which can contain bindings of interest. It is possible to specify a provider URL to select other root contexts.

Selecting the initial root context with a CORBA object URL

There are several object keys registered with the bootstrap server that you can use to select the root context for the initial context. To select a particular root context with a CORBA object URL object key, set the object key to the corresponding value. The default object key is NameService. Using JNDI yields the server root context. A table that lists the different root contexts and their corresponding object key follows:

Root Context	CORBA Object URL Object Key
Server Root	NameServiceServerRoot
Cell Persistent Root	NameServiceCellPersistentRoot
Cell Root	NameServiceCellRoot
Node Root	NameServiceNodeRoot

The following example shows the use of a corbaloc URL with the object key set to select the cell persistent root context as the initial context.

```

...
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
...
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL,
        "corbaloc:iiop:myhost.mycompany.com:2809/NameServiceCellPersistentRoot");
Context initialContext = new InitialContext(env);
...

```

Selecting the initial root context with the name space root property

You can also select the initial root context by passing a name space root property setting to the InitialContext constructor. Generally, the object key setting described above is sufficient. Sometimes a property setting is preferable. For example, you can set the root context property on the Java invocation to make which server root is being used as the initial context transparent to the application. The default server root property setting is defaultroot, which yields the server root context.

Root Context	Name Space Root Property Value
Server Root	bootstrapservroot

Root Context	Name Space Root Property Value
Cell Persistent Root	cellpersistentroot
Cell Root	cellroot
Node Root	bootstrapnoderoot

The initial context factory ignores the name space root property if the provider URL contains an object key other than NameService.

The following example shows use of the name space root property to select the cell persistent root context as the initial context. Note that available constants are used instead of hard-coding the property name and value.

```

...
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
import com.ibm.websphere.naming.PROPS;
...
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, "corbaloc:iiop:myhost.mycompany.com:2809");
env.put(PROPS.NAME_SPACE_ROOT, PROPS.NAME_SPACE_ROOT_CELL_PERSISTENT);
Context initialContext = new InitialContext(env);
...

```

Example: Looking up an EJB home with JNDI

Most applications which use JNDI run in a container. Some do not. The name used to look up an object depends on whether or not the application is running in a container. The examples below show lookups from each type of application. Sometimes it is more convenient for an application to use a corbaname URL as the lookup name. Container-based JNDI clients and thin Java clients can use a corbaname URL. An example of a lookup with a corbaname URL is also included in this section.

JNDI lookup from an application running in a container

Applications that run in a container can use `java:` lookup names. Lookup names of this form provide a level of indirection such that the lookup name used to look up an object is not dependent on the object's name as it is bound in the name server's name space. The deployment descriptors for the application provide the mapping from the `java:` name and the name server lookup name. The container sets up the `java:` name space based on the deployment descriptor information so that the `java:` name is correctly mapped to the corresponding object.

The following example shows a lookup of an EJB home. The actual home lookup name is determined by the application's deployment descriptors.

```

// Get the initial context as shown in a previous example
...
// Look up the home interface using the JNDI name
try {
    java.lang.Object ejbHome =
        initialContext.lookup(
            "java:comp/env/com/mycompany/accounting/AccountEJB");
    accountHome = (AccountHome)javax.rmi.PortableRemoteObject.narrow(
        (org.omg.CORBA.Object) ejbHome, AccountHome.class);
}

```

```

}
  catch (NamingException e) { // Error getting the home interface
    ...
  }
}

```

JNDI lookup from an application that does not run in a container

Applications that do not run in a container cannot use `java:` lookup names because it is the container which sets the `java:` name space up for the application. Instead, an application of this type must look the object up directly from the name server. Each application server contains a name server. System artifacts such as EJB homes are bound relative to the server root context in that name server. The various name servers are federated by means of a system name space structure. The recommended way to look up objects on different servers is to qualify the name so that the name resolves from any initial context in the cell. If a relative name is used, the initial context must be the same server root context as the one under which the object is bound. The form of the qualified name depends on whether the qualified name is a topology-based name or a fixed name. A topology based name depends on whether the object resides in a single server or a server cluster. Examples of each form of qualified name follow.

- **Topology-based qualified names**

Topology-based qualified names traverse through the system name space to the server root context context under which the target object is bound. A topology-based qualified name resolves from any initial context in the cell. The topology-based qualified name depends on whether the object resides on a single server or server cluster. Examples of each lookup follow.

Single server

The following example shows a lookup of an EJB home that is running in the single server, `MyServer`, configured in the node, `Node1`.

```

// Get the initial context as shown in a previous example
// Using the form of lookup name below, it doesn't matter which
// server in the cell is used to obtain the initial context.
...
// Look up the home interface using the JNDI name
try {
    java.lang.Object ejbHome = initialContext.lookup(
        "cell/nodes/Node1/servers/MyServer/com/mycompany/accounting/AccountEJB");
    accountHome = (AccountHome)javax.rmi.PortableRemoteObject.narrow(
        (org.omg.CORBA.Object) ejbHome, AccountHome.class);
}
catch (NamingException e) { // Error getting the home interface
    ...
}

```

Server cluster

The example below shows a lookup of an EJB home which is running in the cluster, `MyCluster`. The name can be resolved if any of the cluster members is running.

```

// Get the initial context as shown in a previous example
// Using the form of lookup name below, it doesn't matter which
// server in the cell is used to obtain the initial context.
...
// Look up the home interface using the JNDI name
try {
    java.lang.Object ejbHome = initialContext.lookup(
        "cell/clusters/MyCluster/com/mycompany/accounting/AccountEJB");
    accountHome = (AccountHome)javax.rmi.PortableRemoteObject.narrow(
        (org.omg.CORBA.Object) ejbHome, AccountHome.class);
}

```

```

    }
    catch (NamingException e) { // Error getting the home interface
        ...
    }
}

```

- **Fixed qualified names**

If the target object has a cell-scoped fixed name defined for it, you can use its qualified form instead of the topology-based qualified name. Even though the topology-based name works, the fixed name does not change with the specific cell topology or with the movement of the target object to a different server. An example lookup with a qualified fixed name is shown below.

```

// Get the initial context as shown in a previous example
// Using the form of lookup name below, it doesn't matter which
// server in the cell is used to obtain the initial context.
...
// Look up the home interface using the JNDI name
try {
    java.lang.Object ejbHome = initialContext.lookup(
        "cell/persistent/com/mycompany/accounting/AccountEJB");
    accountHome = (AccountHome)javax.rmi.PortableRemoteObject.narrow(
        (org.omg.CORBA.Object) ejbHome, AccountHome.class);
}
catch (NamingException e) { // Error getting the home interface
    ...
}

```

JNDI lookup with a corbaname URL

A corbaname can be useful at times as a lookup name. If, for example, the target object is not a member of the federated name space and cannot be located with a qualified name, a corbaname can be a convenient way to look up the object. A lookup with a corbaname URL follows.

```

// Get the initial context as shown in a previous example
...
// Look up the home interface using a corbaname URL
try {
    java.lang.Object ejbHome = initialContext.lookup(
        "corbaname:iiop:someHost:2809#com/mycompany/accounting/AccountEJB");
    accountHome = (AccountHome)javax.rmi.PortableRemoteObject.narrow(
        (org.omg.CORBA.Object) ejbHome, AccountHome.class);
}
catch (NamingException e) { // Error getting the home interface
    ...
}

```

Example: Looking up a JavaMail session with JNDI

The example below shows a lookup of a JavaMail resource. The actual lookup name is determined by the application's deployment descriptors.

```

// Get the initial context as shown above
...
Session session =
    (Session) initialContext.lookup("java:comp/env/mail/MailSession");

```

JNDI interoperability considerations

This section explains considerations to take into account when interoperating with previous releases of WebSphere Application Server and with non-WebSphere Application Server JNDI clients. Also, the way resources from MQSeries must be bound to the name space has changed and is described below.

Interoperability with previous WebSphere Application Server Releases

- **EJB clients running on WebSphere Application Server V3.5 or V4.0 accessing EJB applications running on WebSphere Application Server V5**

Applications migrated from previous versions of WebSphere Application Server may still have clients still running in a previous release. The default initial JNDI context for EJB clients running on previous versions of WebSphere Application Server is the cell persistent root (legacy root). The home for an enterprise bean deployed in version 5 is bound to its server's server root context. In order for the EJB lookup name for down-level clients to remain unchanged, configure a binding for the EJB home under the cell persistent root.

Note: EJB clients running in version 3.5 must be running in version 3.5.5 or above, or in version 3.5.3 or 3.5.4 with e-fix PQ51387 installed.

- **EJB clients running on WebSphere Application Server V5 accessing EJB applications running on WebSphere Application Server V3.5 or V4.0 servers**

The default initial context for a WebSphere Application Server v3.5 or v4.0 server is the correct initial context. Simply look up the JNDI name under which the EJB home is bound.

Note: To enable WebSphere Application Server V5 clients to access version 3.5.x and 4.0.x servers, the down-level installations must have e-fix PQ60074 installed.

EJB clients running in an environment other than WebSphere Application Server accessing EJB applications running on WebSphere Application Server V5 servers

When an EJB application running in WebSphere Application Server V5 is accessed by a non-WebSphere Application Server EJB client, the JNDI initial context factory is presumed to be a non-WebSphere Application Server implementation. In this case, the default initial context will be the cell root. If the JNDI service provider being used supports CORBA object URLs, the corbaname format can be used to look up the EJB home. The construction of the stringified name depends on whether the object is installed on a single server or cluster, as shown below.

- **Single server**

```
initialContext.lookup(
    "corbaname:iiop:myHost:2809#cell/nodes/node1/servers/server1/myEJB");
```

According to the URL above, the bootstrap host and port are myHost and 2809, and the enterprise bean is installed in a server **server1** in node **node1** and bound in that server under the name **myEJB**.

- **Server cluster**

```
initialContext.lookup(
    "corbaname:iiop:myHost:2809#cell/clusters/myCluster/myEJB");
```

According to the URL above, the bootstrap host and port are **myHost** and **2809**, and the enterprise bean is installed in a server cluster named **myCluster** and bound in that cluster under the name **myEJB**.

The above lookup will work with any name server bootstrap host and port configured in the same cell.

The above lookup will also work if the bootstrap host and port belongs to a member of the cluster itself. To avoid a single point of failure, the bootstrap server host and port for each cluster member could be listed in the URL as follows:


```
initialContext.lookup(  
    "corbaname:iop:host1:9810,host2:9810#cell/clusters/myCluster/myEJB");
```

The name prefix **cell/clusters/myCluster/** is not necessary if bootstrapping to the cluster itself, but it will work. The prefix is needed, however, when looking up enterprise beans in other clusters. Name bindings under the **clusters** context are implemented on the name server to resolve to the server root of a running cluster member during a lookup; thus avoiding a single point of failure.

- **Without CORBA object URL support**

If the JNDI initial context factory being used does not support CORBA object URLs, the initial context can be obtained from the server, and the lookup can be performed on the initial context as follows:

```
Hashtable env = new Hashtable();  
env.put(CONTEXT.PROVIDER_URL, "iop://myHost:2809");  
Context ic = new InitialContext(env);  
Object o = ic.lookup("cell/clusters/myCluster/myEJB");
```

Binding resources from MQSeries 5.2

In previous releases of WebSphere Application Server, the MQSeries jmsadmin tool could be used bind resources to the name space. When used with a WebSphere Application Server V5 name space, the resource will be bound within a transient partition in the name space and will not persist past the life of the server process. Instead of binding the MQSeries resources with the jmsadmin tool, bind them from the WebSphere Application Server administrative console, under Resources in the left panel on the console

JNDI caching

To increase the performance of JNDI operations, the WebSphere Application Server JNDI implementation employs caching to reduce the number of remote calls to the name server for lookup operations. For most cases, use the default cache setting.

When an InitialContext object is instantiated, an association is established between the InitialContext instance and a cache. The initial context and any contexts returned directly or indirectly from a lookup on the initial context are all associated with that same cache instance. By default, the association is based on the provider URL, in particular, the host name and port. The caller can specify the cache name to override this default behavior. A cache instance of a given name is shared by all instances of InitialContext configured to use a cache of that name which were created with the same context class loader in effect. Two EJB applications running in the same server will use their own cache instances, if they are using different context class loaders, even if the cache names are the same.

After an association between an InitialContext instance and cache is established, the association does not change. A `javax.naming.Context` object returned from a lookup operation inherits the cache association of the Context object on which the lookup was performed. Changing cache property values with the `Context.addToEnvironment()` or `Context.removeFromEnvironment()` method does not affect cache behavior. You can change properties affecting a given cache instance with each InitialContext instantiation.

A cache is restricted to a process and does not persist past the life of that process. A cached object is returned from lookup operations until either the max cache life for the cache is reached, or the max entry life for the object's cache entry is reached.

After this time, a lookup on the object causes the cache entry for the object to be refreshed. If a bind or rebind operation is executed on an object, the change is not reflected in any caches other than the one associated with the context from which the bind or rebind was issued. This scenario is most likely to happen when multiple processes are involved, since different processes do not share the same cache, and context objects in all threads in a process typically share the same cache instance for a given name service provider.

Usually, cached objects are relatively static entities, and objects becoming stale are not a problem. However, you can set timeout values on cache entries or on a cache so that cache contents are periodically refreshed.

JNDI cache settings

Various cache property settings follow. Ensure that all property values are string values.

com.ibm.websphere.naming.jndicache.cachename

The name of the cache to associate with an initial context instance can be specified with this property.

It is possible to create multiple InitialContext instances, each operating on the name space of a different name server. By default, objects from each bootstrap address are cached separately, since they each involve independent name spaces and name collisions could occur if they used the same cache. The provider URL specified when the initial context is created by default serves as the basis for the cache name. With this property, a JNDI client can specify a cache name. Valid options for cache names follow:

Valid options	Resulting cache behavior
providerURL (default)	Use the value for java.naming.provider.url property as the basis for the cache name. Cache names are based on the bootstrap host and port specified in the URL. The bootstrap host is normalized to a fully qualified name, if possible. For example, "corbaname:iiop:server1:2809#some/starting/context" and "corbaloc:iiop://server1" are normalized to the same cache name. If no provider URL is specified, a default cache name is used.
Any string	Use the specified string as the cache name. You can use any arbitrary string with a value other than "providerURL" as a cache name.

com.ibm.websphere.naming.jndicache.cacheobject

Turn caching on or off and clear an existing cache with this property.

By default, when an InitialContext is instantiated, it is associated with an existing cache or, if one does not exist, a new one is created. An existing cache is used with its existing contents. In some circumstances, this behavior is not desirable. For example, when objects that are looked up change frequently, they can become stale in the cache. Other options are available. The following table lists these other options along with the corresponding property value.

Valid values	Resulting cache behavior
--------------	--------------------------

populated (default)	Use a cache with the specified name. If the cache already exists, leave existing cache entries in the cache; otherwise, create a new cache.
cleared	Use a cache with the specified name. If the cache already exists, clear all cache entries from the cache; otherwise, create a new cache.
none	Do not cache. If this option is specified, the cache name is irrelevant. Therefore, this option will not disable a cache that is already associated with other InitialContext instances. The InitialContext that is instantiated is not associated with any cache.

com.ibm.websphere.naming.jndicache.maxcachelife

Impose a limit to the age of a cache with this property.

By default, cached objects remain in the cache for the life of the process or until cleared with the `com.ibm.websphere.naming.jndicache.cacheobject` property set to "cleared". This property enables a JNDI client to set the maximum life of a cache. This property differs from the `maxentrylife` property (below) in that the entire cache is cleared when the cache lifetime is reached. The table below lists the various `maxcachelife` values and their affect on cache behavior:

Valid options	Resulting cache behavior
0 (default)	Make the cache lifetime unlimited.
Positive integer	Set the maximum lifetime of the entire cache, in minutes, to the specified value. When the maximum lifetime for the cache is reached, the next attempt to read any entry from the cache causes the cache to be cleared

com.ibm.websphere.naming.jndicache.maxentrylife

Impose a limit to the age of individual cache entries with this property.

By default, cached objects remain in the cache for the life of the process or until cleared with the `com.ibm.websphere.naming.jndicache.cacheobject` property set to `cleared`. This property enables a JNDI client to set the maximum lifetime of individual cache entries. This property differs from the `maxcachelife` property in that individual entries are refreshed individually as their maximum lifetime reached. This might avoid any noticeable change in performance that might occur if the whole cache is cleared at once. The table below lists the various `maxentrylife` values and their effect on cache behavior:

Valid options	Resulting cache behavior
0 (default)	Lifetime of cache entries is unlimited.
Positive integer	Set the maximum lifetime of individual cache entries, in minutes, to the specified value. When the maximum lifetime for an entry is reached, the next attempt to read the entry from the cache causes the individual cache entry to refresh.

Example: Controlling JNDI cache behavior from a program

Following are examples that illustrate how you can use JNDI cache properties to achieve the desired cache behavior. Cache properties take effect when an `InitialContext` object is constructed.

```
import java.util.Hashtable;
import javax.naming.InitialContext;
import javax.naming.Context;
import com.ibm.websphere.naming.PROPS;

/*****
 Caching discussed in this section pertains to the WebSphere Application Server
 initial context factory. Assume the property, java.naming.factory.initial, is set
 to "com.ibm.websphere.naming.WsnInitialContextFactory" as a java.lang.System property.
 *****/

Hashtable env;
Context ctx;

// To clear a cache:

env = new Hashtable();
env.put(PROPS.JNDI_CACHE_OBJECT, PROPS.JNDI_CACHE_OBJECT_CLEARED);
ctx = new InitialContext(env);

// To set a cache's maximum cache lifetime to 60 minutes:

env = new Hashtable();
env.put(PROPS.JNDI_CACHE_MAX_LIFE, "60");
ctx = new InitialContext(env);

// To turn caching off:

env = new Hashtable();
env.put(PROPS.JNDI_CACHE_OBJECT, PROPS.JNDI_CACHE_OBJECT_NONE);
ctx = new InitialContext(env);

// To use caching and no caching:

env = new Hashtable();
env.put(PROPS.JNDI_CACHE_OBJECT, PROPS.JNDI_CACHE_OBJECT_POPULATED);
ctx = new InitialContext(env);
env.put(PROPS.JNDI_CACHE_OBJECT, PROPS.JNDI_CACHE_OBJECT_NONE);
Context noCacheCtx = new InitialContext(env);

Object o;

// Use caching to look up home, since the home should rarely change.
o = ctx.lookup("com/mycom/MyEJBHome");
// Narrow, etc. ...

// Do not use cache if data is volatile.
o = noCacheCtx.lookup("com/mycom/VolatileObject");
// ...
```

JNDI name syntax

JNDI name syntax is the default syntax and is suitable for typical JNDI clients.

This syntax includes the following special characters: forward slash (/) and backslash (\). Components in a name are delimited by a forward slash. The backslash is used as the escape character. A forward slash is interpreted literally if it is escaped, that is, preceded by a backslash. Similarly, a backslash is interpreted literally if it is escaped.

INS name syntax

INS syntax is designed for JNDI clients that need to interoperate with CORBA applications.

The INS syntax allows a JNDI client to make the proper mapping to and from a CORBA name. INS syntax is very similar to the JNDI syntax with the additional special character, dot (.). Dots are used to delimit the `id` and `kind` fields in a name component. A dot is interpreted literally when it is escaped. Only one unescaped dot is allowed in a name component. A name component with a non-empty `id` field and empty `kind` field is represented with only the `id` field value and must not end with an unescaped dot. An empty name component (empty `id` and empty `kind` field) is represented with a single unescaped dot. An empty string is not a valid name component representation.

JNDI to CORBA name mapping considerations

WebSphere Application Server name servers are an implementation of the CORBA CosNaming interface. WebSphere Application Server provides a JNDI implementation which you can use to access CosNaming name servers through the JNDI interface. Issues can exist when mapping JNDI name strings to and from CORBA names.

Each component in a CORBA name consists of an `id` and `kind` field, but a JNDI name component consists of no such fields. Each component in a JNDI name is atomic. Typical JNDI clients do not need to make a distinction between the `id` and `kind` fields of a name component, or know how JNDI name strings map to CORBA names. JNDI clients of this sort can use the JNDI syntax described below. When a name is parsed according to JNDI syntax, each name component is mapped to the `id` field of the corresponding CORBA name component. The `kind` field always has an empty value. This basic syntax is the least obtrusive to the JNDI client in that it has the fewest special characters. However, you cannot represent with this syntax a CORBA name with a non-empty `kind` field. This restriction can prevent EJB applications from interoperating with CORBA applications.

Some clients, however must interoperate with CORBA applications which use CORBA names with non-empty `kind` fields. These JNDI clients must make a distinction between `id` and `kind` so that JNDI names are correctly mapped to CORBA names, particularly when the CORBA names contain components with non-null `kind` fields. Such JNDI clients can use the INS name syntax. With its additional special character, you can use INS to represent any CORBA name. Use of this syntax is not recommended unless it is necessary, because this syntax is more restrictive from the JNDI client's perspective in that the JNDI client must be aware that name components with multiple unescaped dots are syntactically invalid. INS name syntax is part of the OMG CosNaming Interoperable Naming Specification.

Example: Setting the syntax used to parse name strings

JNDI clients which must interoperate with CORBA applications may need to use INS name syntax to represent names in string format. The name syntax property may be passed to the `InitialContext` constructor through its parameter, in the System properties, or in a `jndi.properties` file. The initial context and any contexts looked up from that initial context will parse name strings based on the specified syntax.

The following example shows how to set the name syntax to make the initial context parse name strings according to INS syntax.

```

...
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
import com.ibm.websphere.naming.PROPS; // WebSphere naming constants
...
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, ...);
env.put(PROPS.NAME_SYNTAX, PROPS.NAME_SYNTAX_INS);
Context initialContext = new InitialContext(env);
// The following name maps to a CORBA name component as follows:
//   id = "a.name", kind = "in.INS.format"
// The unescaped dot is used as the delimiter.
// Escaped dots are interpreted literally.
java.lang.Object o = initialContext.lookup("a\\.name.in\\.INS\\.format");
...

```

Developing applications that use CosNaming (CORBA Naming interface)

CORBA clients can perform naming operations on WebSphere name servers through the CosNaming interface. The following examples show how to obtain an ORB instance and an initial context as well as how to look up an EJB home.

Note: To enable WebSphere Application Server V5 clients to access Versions 3.5.x and 4.0.x servers, the earlier installations must have e-fix PQ60074 installed.

1. Get an initial context
2. Perform desired CosNaming operations

Example: Getting an initial context with CosNaming

In the WebSphere Application Server, an initial context is obtained from a bootstrap server. The address for the bootstrap server consists of a host and port. To get an initial context, you must know the host and port for the server that is used as the bootstrap server.

Obtaining an initial context consists of two basic steps:

1. Obtain an ORB reference
2. Invoke a method on the ORB to obtain the initial reference

These steps are now explained in more detail.

Obtaining an ORB reference

Pure CosNaming clients, that is clients that are not running in a server process, must create and initialize an ORB instance with which to obtain the initial context. CosNaming clients which run in server processes can obtain a reference to the server ORB with a JNDI lookup. The following examples illustrate how to create and initialize a client ORB and how to obtain a server ORB reference.

Creating a client ORB instance

To create an ORB instance, invoke the static method, `org.omg.CORBA.ORB.init`. The `init` method requires a property set to the name of the ORB class you want to instantiate. An ORB implementation with the class name

com.ibm.CORBA.iiop.ORB is included with the WebSphere Application Server. The WebSphere Application Server ORB recognizes additional properties with which you can specify initial references.

The basic steps for creating an ORB are as follows:

1. Create a Properties object.
2. Set the ORB class property to WebSphere Application Server's ORB class.
3. If the bootstrap server is INS-compliant, set the initial reference properties. If the bootstrap server is not INS-compliant (meaning, WebSphere Application Server v4.0.x or earlier), set bootstrap host and port for bootstrap server.
4. Invoke ORB.init, passing in the Properties object.

Usage scenario

```
...
import java.util.Properties;
import org.omg.CORBA.ORB;
...
Properties props = new Properties();
props.put("org.omg.CORBA.ORBClass", "com.ibm.CORBA.iiop.ORB");
props.put("com.ibm.CORBA.ORBInitRef.NameService",
    "corbaloc:iiop:myhost.mycompany.com:2809/NameService");
props.put("com.ibm.CORBA.ORBInitRef.NameServiceServerRoot",
    "corbaloc:iiop:myhost.mycompany.com:2809/NameServiceServerRoot");
// props.put("com.ibm.CORBA.BootstrapHost", "myhost.mycompany.com");
// Use this if bootstrap server is WebSphere 4.0.x or before
// props.put("com.ibm.CORBA.BootstrapPort", "2809");
// Use this if bootstrap server is WebSphere 4.0.x or before
ORB _orb = ORB.init((String[])null, props);
...

...
import java.util.Properties;
import org.omg.CORBA.ORB;
...
Properties props = new Properties();
props.put("org.omg.CORBA.ORBClass", "com.ibm.ws390.orb.ORB");
props.put("com.ibm.CORBA.ORBInitRef.NameService",
    "corbaloc:iiop:myhost.mycompany.com:2809/NameService");
props.put("com.ibm.CORBA.ORBInitRef.NameServiceServerRoot",
    "corbaloc:iiop:myhost.mycompany.com:2809/NameServiceServerRoot");
// props.put("com.ibm.CORBA.BootstrapHost", "myhost.mycompany.com");
// Use this if bootstrap server is WebSphere 4.0.x or before
// props.put("com.ibm.CORBA.BootstrapPort", "2809");
// Use this if bootstrap server is WebSphere 4.0.x or before
ORB _orb = ORB.init((String[])null, props);
...
```

Notice the initial reference definitions for NameService and NameServiceServerRoot. The initial context returned for NameService depends on the type of bootstrap server. The key NameServiceServerRoot is a key introduced in WebSphere Application Server V5. For more information on initial contexts, see the section Initial Contexts.

Note: The properties com.ibm.CORBA.BootstrapHost and com.ibm.CORBA.BootstrapPort are deprecated. They are needed, however, to connect to WebSphere Application Servers of Version 4.0.x or earlier. The default bootstrap host is the local host and the default port is 2809.

Obtaining a reference to the server ORB

CosNaming clients which run in a server process can obtain a reference to the server ORB with a JNDI lookup on a java: name, shown as follows:

Usage scenario

```
...
import javax.naming.Context;
import javax.naming.InitialContext;
import org.omg.CORBA.ORB;
...
Context initialContext = new InitialContext();
ORB orb = (ORB) initialContext.lookup("java:comp/ORB");
...
```

Using an ORB reference to get an initial naming reference

There are two basic ways to get an initial CosNaming context. Both ways involve an ORB method invocation. The first way is to invoke the `resolve_initial_references` method on the ORB with an initial reference key. For this call to work, the ORB must be initialized with an initial reference for that key. The other way is to invoke the `string_to_object` method on the ORB, passing in a CORBA object URL with the host and port of the bootstrap server. The following examples illustrate both approaches.

Invoking `resolve_initial_references`

Once an ORB reference is obtained, invoke the `resolve_initial_references` method on the ORB to obtain a reference to the initial context. The following code example invokes `resolve_initial_reference` on an ORB reference.

Usage scenario

```
...
import org.omg.CORBA.ORB;
import org.omg.CosNaming.NamingContextExt;
import org.omg.CosNaming.NamingContextExtHelper;
...
// Obtain ORB reference as shown in examples earlier in this section
...
org.omg.CORBA.Object obj = _orb.resolve_initial_references("NameService");
NamingContextExt initCtx = NamingContextExtHelper.narrow(obj);
...
```

Note that the key `NameService` is passed to the `resolve_initial_references` method. Other initial context keys are registered in WebSphere Application Servers. For example, `NameServiceServerRoot` can be used to obtain a reference to the server root context in the bootstrap name server. For more information on the initial contexts registered in server ORBs, please see the section `Initial Contexts`.

Invoking `string_to_object` with a CORBA object URL

You can use an INS-compliant ORB to obtain an initial context even if the ORB is not initialized with any initial references or bootstrap properties, or if those property settings are for a different server than the name server from which you want to obtain the initial context. To obtain an initial context by explicitly specifying the bootstrap name server, invoke the `string_to_object` method on the ORB, passing in a CORBA object URL which contains the bootstrap server host and port.

The code in the example below invokes the `string_to_object` method on an existing ORB reference, passing in a CORBA object URL which identifies the desired initial context.

Usage scenario


```

...
import org.omg.CORBA.ORB;
import org.omg.CosNaming.NamingContextExt;
import org.omg.CosNaming.NamingContextExtHelper;
...
// Obtain ORB reference as shown in examples earlier in this section
...
org.omg.CORBA.Object obj =
    orb.string_to_object("corbaloc:iiop:myhost.mycompany.com:2809/NameService");
NamingContextExt initCtx = NamingContextExtHelper.narrow(obj);
...

```

Note that the key `NameService` is used in the `corbaloc` URL. Other initial context keys are registered in WebSphere Application Servers. For example, you can use `NameServiceServerRoot` to obtain a reference to the server root context in the bootstrap name server.

Using an existing ORB and invoking `string_to_object` with a CORBA object URL with multiple name server addresses to get an initial context

CORBA object URLs can contain more than one bootstrap server address. Use this feature when attempting to obtain an initial context from a server cluster. You can specify the bootstrap server addresses for all servers in the cluster in the URL. The operation will succeed if at least one of the servers is running, eliminating a single point of failure. There is no guarantee of any particular order in which the address list will be processed. For example, the second bootstrap server address may be used to obtain the initial context even though the first bootstrap server in the list is available. An example of a `corbaloc` URL with multiple addresses follows.

```

...
import org.omg.CORBA.ORB;
import org.omg.CosNaming.NamingContextExt;
import org.omg.CosNaming.NamingContextExtHelper;
...
// Assume orb is an existing ORB instance
org.omg.CORBA.Object obj = orb.string_to_object(
    "corbaloc::myhost1:9810,:myhost1:9811,:myhost2:9810/NameService");
NamingContextExt initCtx = NamingContextExtHelper.narrow(obj);
...

```

Example: Looking up an EJB home with `CosNaming`

You can look up an EJB home or other CORBA object from a WebSphere Application Server name server through the CORBA `CosNaming` interface. You can invoke `resolve` or `resolve_str` on the initial context, or you can invoke `string_to_object` on the ORB. You can use a qualified name so that the name resolves regardless of which name server the lookup is executed on, or use an unqualified name that only resolves from the server root context on the name server that actually contains the object binding. (The qualified name traverses the federated system name space to the specified server root context.)

Qualified and unqualified names

Each application server contains a name server. System artifacts such as EJB homes are bound in that name server. The various name servers are federated by means of a system name space structure. The recommended way to look up objects on different servers is to use a qualified name. A qualified name can be a topology-based name, based on the name of the cluster or single server and node that contains the object. You can define fixed qualified names for objects. With qualified names, you can look up objects residing on different servers from the same initial context by traversing the system name space structure. Alternatively,

you can use an unqualified name, but an unqualified name will only resolve using the name server associated with the object's application server.

CosNaming.resolve (and resolve_str) vs. ORB.string_to_object

If you have an initial context from any name server in a WebSphere Application Server cell, you can look up any CORBA object with a qualified name. You do not need additional host and port information for the target object's name server.

Alternatively, you can look up an object by invoking `string_to_object` on the ORB, passing in a corbaname URL. Typically, an IIOP type URL is specified, so the bootstrap address information required for an initial context must be contained in the URL. You can use a qualified or unqualified stringified name, but an unqualified name resolves only if the initial context is from the name server in which the object is bound.

The following examples show CosNaming resolve operations using qualified topology-based lookup names and an unqualified lookup name.

CosNaming resolve operation using a qualified name

The topology-based qualified name for an object depends on whether the object is bound in a single server or a server cluster. Examples of each follow.

Single Server

The following example shows the lookup of an EJB home that is running in a single server. The enterprise bean that is being looked up is running in the server, `MyServer`, on the node, `Node1`.

```
// Get the initial context as shown in the previous example
// Using the form of lookup name below, it doesn't matter which
// server in the cell is used to obtain the initial context.
...
// Look up the home interface using the name under which the EJB home is bound
org.omg.CORBA.Object ejbHome = initialContext.resolve_str(
    "cell/nodes/Node1/servers/MyServer/mycompany/accounting/AccountEJB");
accountHome =
    (AccountHome)javax.rmi.PortableRemoteObject.narrow(ejbHome, AccountHome.class);
```

Server Cluster

The following example shows a lookup of an EJB home that is running in a cluster. The enterprise bean being that is looked up is running in the cluster, `Cluster1`. The name can be resolved if any of the cluster members is running.

Usage scenario

```
// Get the initial context as shown in the previous example
// Using the form of lookup name below, it doesn't matter which
// server in the cell is used to obtain the initial context.
...
// Look up the home interface using the name under which the EJB home is bound
org.omg.CORBA.Object ejbHome = initialContext.resolve_str(
    "cell/clusters/Cluster1/mycompany/accounting/AccountEJB");
accountHome =
    (AccountHome)javax.rmi.PortableRemoteObject.narrow(ejbHome, AccountHome.class);
```

ORB string_to_object operation using an unqualified stringified name

If the resolve operation is being performed on the name server that contains the object, the system name space does not need to be traversed, and you can use an

unqualified lookup name. Note that this name does not resolve on other name servers. If an unqualified name is provided, the object key must be `NameServiceServerRoot` so that the correct initial context is selected. If a qualified name is provided, you can use the default key of `NameService`.

The following example shows a lookup of an EJB home. The enterprise bean that is being looked up is bound on the name server running on the host `myHost` on port 2809. Note the object key of `NameServiceServerRoot`.

```
// Assume orb is an existing ORB instance
...
// Look up the home interface using the name under which the EJB home is bound
org.omg.CORBA.Object ejbHome = orb.string_to_object(
    "corbaname:iiop:myHost:2809/NameServiceServerRoot#mycompany/accounting");
AccountHome =
    (AccountHome)javax.rmi.PortableRemoteObject.narrow(ejbHome, AccountHome.class);
```

Configured name bindings

Administrators can configure bindings into the name space. A configured binding is different from a programmatic binding in that the system creates the binding every time a server is started, even if the target context is in a transient partition.

Administrators can add name bindings to the name space through the configuration. Name servers add these configured bindings to the name space view, by reading the configuration data for the bindings. Configuring bindings is an alternative to creating the bindings from a program. Configured bindings have the advantage of being created each time a server starts, even when the binding is created in a transient partition of the name space. Cell-scoped configured bindings provide interoperability with JNDI clients running on previous versions of WebSphere Application Server. Additionally, you can configure cell-scoped bindings to create a fixed qualified name for server objects.

Scope

You can configure a binding at one of the following three scopes: cell, node, or server. Cell-scoped bindings are created under the cell persistent root context. Node-scoped bindings are created under the node persistent root context for the specified node. Server-scoped bindings are created under the server root context for the selected server. If the target server of a server-scoped binding is a cluster, the binding is created under the server root context of each cluster member.

Note: The term *server* includes clusters and can be used interchangeably with the term *cluster* with respect to configured bindings. When applied to a cluster, a server-scoped binding is created in the server root for all member servers.

The scope you select for new bindings depends on how the binding is to be used. For example, if the binding is not specific to any particular node or server, or if you do not want the binding to be associated with any specific node or server, a cell-scoped binding is a suitable scope. Defining fixed names for enterprise beans to create fixed qualified names is just such an application. If a binding is to be used only by clients of an application running on a particular server, or if you want to configure a binding with the same name on different servers which resolve to different objects, a server-scoped binding would be appropriate. Note that two servers can have configured bindings with the same name but resolve to different objects. At the cell scope, only one binding with a given name can exist.

Intermediate Contexts

Intermediate contexts created with configured bindings are read-only. For example, if an EJB home binding is configured with the name `some/compound/name/ejbHome`, the intermediate contexts `some`, `some/compound`, and `some/compound/name` will be created as read-only contexts. You cannot add, update, or remove any read-only bindings.

The configured binding name cannot conflict with existing bindings. However, configured bindings can use the same intermediate context names. Therefore, a configured binding with the name `some/compound/name2/ejbHome2` does not conflict with the previous example name.

Configured binding types

Types of objects that you can bind follow:

EJB: EJB home installed in some server in the cell

The following data is required to configure an EJB home binding:

- JNDI name of the EJB server or server cluster where the enterprise bean is deployed
- Target root for the configured binding (scope)
- The name of the configured binding, relative to the target root.

This type of binding is of special significance because you can use it to provide interoperability with WebSphere Application Server v3.5.x and v4.0.x JNDI clients. The default initial context for these earlier clients is the cell persistent root, which is different from the initial context of the server root for WebSphere Application Server V5 JNDI clients. If you migrate an application to the current release, you can configure an EJB binding at the cell scope so that the lookup names for the enterprise bean do not change for clients still running in an earlier WebSphere Application Server version.

A cell-scoped EJB binding is also useful for creating a fixed lookup name for an enterprise bean so that the qualified name is not dependent on the topology.

CORBA: CORBA object available from some CosNaming name server

You can identify any CORBA object bound into some INS compliant CosNaming server with a `corbaname` URL. The referenced object does not have to be available until the binding is actually referenced by some application.

The following data is required in order to configure a CORBA object binding:

- The `corbaname` URL of the CORBA object
- An indicator if the bound object is a context or leaf node object (to set the correct CORBA binding type of context or object).
- Target root for the configured binding
- The name of the configured binding, relative to the target root.

Indirect: Any object bound in WebSphere Application Server name space accessible with JNDI

Besides CORBA objects, this includes `javax.naming.Referenceable`, `javax.naming.Reference`, and `java.io.Serializable` objects. The target object itself is not bound to the name space. Only the information required to look up the object is bound. Therefore, the referenced name server does not have to be running until the binding is actually referenced by some application. The following data is required in order to configure an indirect JNDI lookup binding:

- JNDI provider URL of name server where object resides
- JNDI lookup name of object
- Target root for the configured binding (scope)
- The name of the configured binding, relative to the target root.

A cell-scoped indirect binding is useful when creating a fixed lookup name for a resource so that the qualified name is not dependent on the topology. You can also achieve this topology by widening the scope of the resource definition.

Note: WebSphere Application Server v3.5.x clients cannot access this type of binding .

String: String constant

You can configure a binding of a string constant. The following data is required to configure a string constant binding:

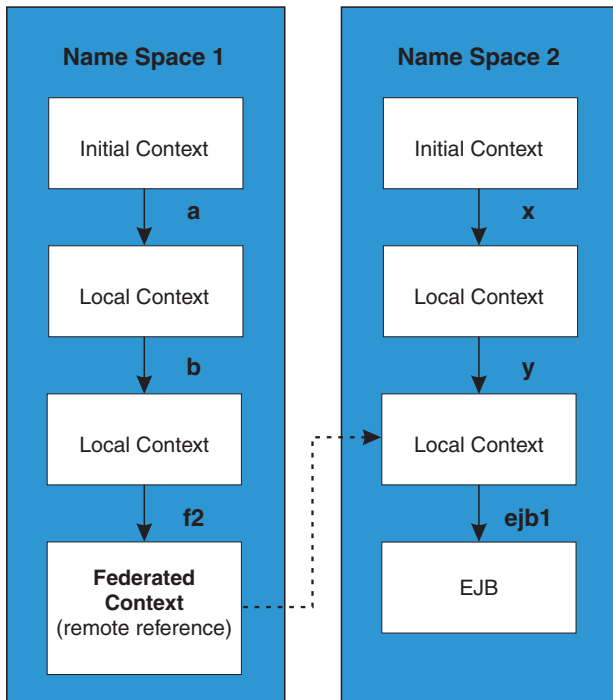
- String constant value
- Target root for the configured binding (scope)
- The name of the configured binding, relative to the target root.

Name space federation

Federating name spaces involves binding contexts from one name space into another name space.

For example, assume that a name space, Name Space 1, contains a context under the name a/b. Also assume that a second name space, Name Space 2, contains a context under the name x/y. (See the following illustration.) If context x/y in Name Space 2 is bound into context a/b in Name Space 1 under the name f2, the two name spaces are federated. Binding f2 is a federated binding because the context associated with that binding comes from another name space. From Name Space 1, a lookup of the name a/b/f2 returns the context bound under the name x/y in Name Space 2. Furthermore, if context x/y contains an Enterprise JavaBeans (EJB) home bound under the name ejb1, the EJB home could be looked up from Name Space 1 with the lookup name a/b/f2/ejb1. Notice that the name crosses name spaces. This fact is transparent to the naming client.

Federated Name Spaces



In a WebSphere Application Server name space, you can create federated bindings with the following restrictions:

- Federation is limited to CosNaming name servers. A WebSphere Application Server name server is a Common Object Request Broker Architecture (CORBA) CosNaming implementation. You can create federated bindings to other CosNaming contexts. You cannot, for example, bind contexts from an LDAP name server implementation.
- If you use JNDI to federate the name space, you must use WebSphere Application Server's initial context factory to obtain the reference to the federated context. If you use some other initial context factory implementation, you either may not be able to create the binding, or the level of transparency may be reduced.
- A federated binding to a non-WebSphere Application Server naming context has the following functional limitations:
 - JNDI operations are restricted to the use of CORBA objects. For example, you can look up EJB homes, but you cannot look up non-CORBA objects such as data sources.
 - JNDI caching is not supported for non-WebSphere Application Server name spaces. This restriction affects the performance of lookup operations only.
- Do not federate two WebSphere Application Server stand-alone server name spaces. Incorrect behavior may result. If you want to federate WebSphere Application Server name spaces, you should use servers running under the Network Deployment or Enterprise packages of WebSphere Application Server.

Name space bindings

Administrators can add name bindings to the name space through the configuration. Name servers add these configured bindings to the name space view by reading the configuration data for the bindings. Configuring bindings is an alternative to creating the bindings from a program.

Configured bindings are created each time a server starts, even when the binding is created in a transient partition of the name space. One major use of configured bindings is to provide interoperability with JNDI clients running on previous versions of the WebSphere Application Server.

There are four different kinds of bindings that you can configure:

- Enterprise JavaBeans (EJB)
- CORBA object
- Indirect Lookup
- String

Configuring and viewing name space bindings

To view or configure an EJB, CORBA, Indirect lookup or string name space binding, complete the following:

1. Open the Administrative console.
2. Click **Environment**.
3. Click **Manage Name Space Bindings**.
4. Select the desired scope by entering in a node name for node-scoped bindings, or a node name and server name for server-scoped bindings, and click **Apply**.
5. To create a new binding, click **New** and follow the instructions. To edit a previously created binding, click the binding you want to edit and proceed to the next step.
6. Edit the Binding identifier, the Name in name space, and the String value fields as desired.

Note: All of these fields are required.
7. Click **Finish** to register the changes.

String binding settings

Use this page to configure a new string binding or to view or edit an existing string binding.

To view this administrative console page, click **Environment > Naming > Name Space Bindings > *string_namespace_binding***.

Scope

Shows the scope of the configured binding. This value indicates the configuration location for the `namebindings.xml` file. This field is for information purposes only and cannot be updated.

If the configured binding is cell-scoped, the starting context is the cell persistent root context. If the configured binding is node-scoped, the starting context is the node persistent root context. If the configured binding is server-scoped, the starting context is the server's server root context.

Binding Type

Shows the type of binding configured. Possible choices are String, EJB, CORBA, and Indirect. This field is for information purposes only and cannot be updated.

Binding Identifier

Specifies the name that uniquely identifies this configured binding.

Name in Name Space

Specifies the name used for this binding in the name space. This name can be a simple or compound name depending on the portion of the name space where this binding is configured.

String Value

Specifies the string to be bound into the name space.

CORBA object binding settings

Use this page to configure a new name binding of a CORBA object binding, or to view or edit an existing CORBA object binding.

To view this administrative console page, click **Environment > Naming > Name Space Bindings > CORBA_namespace_binding**.

Scope

Shows the scope of the configured binding. This value indicates the configuration location for the namebindings.xml file. This field is for information purposes only and cannot be updated.

If the configured binding is cell-scoped, the starting context is the cell persistent root context. If the configured binding is node-scoped, the starting context is the node persistent root context. If the configured binding is server-scoped, the starting context is the server's server root context.

Binding Type

Shows the type of binding configured. Possible choices are String, EJB, CORBA, and Indirect. This field is for information purposes only and cannot be updated.

Binding Identifier

Specifies the name that uniquely identifies this configured binding.

Name in Name Space

Specifies the name used for this binding in the name space. This name can be a simple or compound name depending on the portion of the name space where this binding is configured.

Corbaname URL

Specifies the CORBA name URL string identifying where the object is bound in a CosNaming server.

Federated Context

Specifies whether the target is a CosNaming context (true) or a leaf node object (false).

Value	Result
true	The target object is bound with a context CORBA binding type. If the corbaname URL does not resolve to a NamingContext, an error occurs when the binding is first used (which is when the URL is first resolved).
false	The target object is bound with an object CORBA binding type.

Indirect lookup binding settings

Use this page to configure a new indirect lookup name binding, or to view or edit an existing indirect lookup binding.

To view this administrative console page, click **Environment > Naming > Name Space Bindings** > *indirect_lookup_namespace_binding*.

Scope

Shows the scope of the configured binding. This value indicates the configuration location for the `namebindings.xml` file. This field is for information purposes only and cannot be updated.

If the configured binding is cell-scoped, the starting context is the cell persistent root context. If the configured binding is node-scoped, the starting context is the node persistent root context. If the configured binding is server-scoped, the starting context is the server's server root context.

Binding Type

Shows the type of binding configured. Possible choices are String, EJB, CORBA, and Indirect. This field is for information purposes only and cannot be updated.

Binding Identifier

Specifies the name that uniquely identifies this configured binding.

Name in Name Space

Specifies the name used for this binding in the name space. This name can be a simple or compound name depending on the portion of the name space where this binding is configured.

Provider URL

Specifies the provider URL string needed to obtain a JNDI initial context.

JNDI Name

Specifies the name used to look up the target object from the initial context.

EJB binding settings

Use this page to configure a new EJB binding, or to view or edit an existing EJB binding.

To view this administrative console page, click **Environment > Naming > Name Space Bindings** > *EJB_namespace_binding*.

Scope

Shows the scope of the configured binding. This value indicates the configuration location for the `namebindings.xml` file. This field is for information purposes only and cannot be updated.

If the configured binding is cell-scoped, the starting context is the cell persistent root context. If the configured binding is node-scoped, the starting context is the node persistent root context. If the configured binding is server-scoped, the starting context is the server's server root context.

Binding Type

Shows the type of binding configured. Possible choices are String, EJB, CORBA, and Indirect. This field is for information purposes only and cannot be updated.

Binding Identifier

Specifies the name that uniquely identifies this configured binding.

Name in Name Space

Specifies the name used for this binding in the name space. This name can be a simple or compound name depending on the portion of the name space where this binding is configured.

Enterprise Bean Location

Specifies whether the enterprise bean is running in a server cluster or a single server. If Single Server is specified, type the node name.

Server

Specifies the name of the cluster or non-clustered server in which the enterprise bean is configured.

JNDI Name

Specifies the JNDI name of the deployed enterprise bean (the bean's JNDI name that is in the enterprise bean bindings--not the java:comp name)

Name space binding collection

Use this page to configure a name binding of an EJB, a CORBA CosNaming NamingContext, a CORBA leaf node object, an object that you can look up using JNDI, or a constant string value.

Binding information for configured bindings is stored in the configuration and applied upon startup of the name server for each server within the scope of the binding.

To view the Manage Name Space Bindings Settings page, click **Environment > Naming > Name Space Bindings**.

Click the check boxes to select one or more of the users in your collection. Use the buttons to control the selected users.

Name

Shows the names given to uniquely identify these configured bindings.

Scope

Shows the scope of the configured binding. This value indicates the configuration location for the namebindings.xml file. This field is for information purposes only and cannot be updated.

If the configured binding is cell-scoped, the starting context is the cell persistent root context. If the configured binding is node-scoped, the starting context is the node persistent root context. If the configured binding is server-scoped, the starting context is the server's server root context.

Binding Type

Shows the type of binding configured. Possible choices are String, EJB, CORBA, and Indirect. This field is for information purposes only and cannot be updated.

Configuring name servers

To configure a name server, complete the following:

1. Open the administrative console.
2. Click **Servers**.
3. Click **Application Servers**.
4. Click the application server you want to configure.
5. Click **Server Components**.
6. Click **Name Server**.
7. Edit the fields as desired.

Note: All of these fields are mandatory.

8. To make other changes, click **Custom Properties**.
9. Click **OK** to register your changes.

Name server settings

Use this page to configure Naming Service Provider settings for the application server.

To view this administrative console page, click one of the following paths:

- **Servers > Application Servers > *server_name* > Server Components > Name Server**
- **Servers > JMS Servers > *server_name* > Server Components > Name Server**

Name

Specifies the display name for the server.

Data type	String
-----------	--------

Initial State

Specifies the execution state. The options are: *Started* and *Stopped*.

Data type	String
Default	Started

Troubleshooting name space problems

Many naming problems can be avoided by fully understanding the key underlying concepts of WebSphere Application Server naming.

1. Review the key concepts of WebSphere Application Server naming, especially Name space logical view and Lookup names support in deployment descriptors and thin clients.
2. Review the programming examples that are included in the sections explaining the JNDI and CosNaming interfaces.
3. Read Naming services component troubleshooting tips for additional general information.
4. If you Cannot look up an object hosted by WebSphere Application Server from a servlet, JSP file, or other client, read this article.

dumpNameSpace tool

You can use the dumpNameSpace tool to dump the contents of a name space accessed through a name server. When you invoke the dumpNameSpace tool, the naming service must be active. The dumpNameSpace tool cannot dump name spaces local to the server process, such as those with java: and local: URL schemes. The local: name space contains references to enterprise beans with local interfaces. Use the name space dump utility for java:, local: and server name spaces to dump java: and local: name spaces.

Note that the server root context for the server at the specified host and port is dumped (unless a non-default starting context which precludes it is specified). The server root contexts for other servers are not dumped.

If you run the dumpNameSpace tool, a login prompt is displayed. If you cancel the login prompt, the dumpNameSpace tool continues outbound with an

"UNAUTHENTICATED" credential. Thus, by default, an "UNAUTHENTICATED" credential is used that is equivalent to the "Everyone" access authorization policy. You can modify this default setting by changing the value for the `com.ibm.CSI.performClientAuthenticationRequired` property to true in the `install_dir/properties/sas.client.props` file. When you change this property to true, rerun the `dumpNameSpace` tool, and cancel the login prompt, the authorization fails and the command will not continue outbound.

Command line invocation descriptions of the `dumpNameSpace` tool follow. This section includes sample output.

You can also access this tool through its program interface. Refer to the class `com.ibm.websphere.naming.DumpNameSpace` in the WebSphere Application Server API documentation.

To invoke the tool through the command line, enter the following command from the `WebSphere/AppServer/bin` directory:

Platform	Command
UNIX	<code>dumpNameSpace.sh [[-keyword value]...]</code>
Windows NT	<code>dumpNameSpace [[-keyword value]...]</code>

Parameters

The keywords and associated values for the `dumpNameSpace` utility follow:

-host *myhost.austin.ibm.com*

Indicates the bootstrap host or the WebSphere Application Server host whose name space you want to dump. The value defaults to *localhost*.

-port *nmn*

Indicates the bootstrap port which, if not specified, defaults to **2809**.

-root {*cell* | *server* | *node* | *host* | *legacy* | *tree* | *default*}

Indicates the root context to use as the initial context for the dump. The applicable root options and default root context depend on the type of name server from which the dump is being obtained. This information is provided in the following tables.

For WebSphere Application Servers V5 or later:

<i>cell</i>	DumpNameSpace default. Dump the tree starting at the cell root context.
<i>server</i>	Dump the tree starting at the server root context.
<i>node</i>	Dump the tree starting at the node root context. (Synonymous with <i>host</i> .)

For WebSphere Application Servers v4.0 or later:

<i>legacy</i>	DumpNameSpace default. Dump the tree starting at the legacy root context.
<i>host</i>	Dump the tree starting at the bootstrap host root context. (Synonymous with <i>node</i> .)
<i>tree</i>	Dump the tree starting at the tree root context.

For all WebSphere Application Servers and other name servers:

default	Dump the tree starting at the initial context which JNDI returns by default for that server type. This is the only -root choice that is compatible with WebSphere Application Servers prior to v4.0 and with non-WebSphere Application Server name servers.
---------	---

-url *some provider URL*

Indicates the value for the `java.naming.provider.url` property used to get the initial JNDI context. This option can be used in place of the `-host`, `-port`, and `-root` options. If the `-url` option is specified, the `-host`, `-port`, and `-root` options are ignored.

-factory *com.ibm.websphere.naming.WsnInitialContextFactory*

Indicates the initial context factory to be used to get the JNDI initial context. The value defaults to: `com.ibm.websphere.naming.WsnInitialContextFactory`. The default value generally does not need to be changed.

-startAt *some/subcontext/in/the/tree*

Indicates the path from the bootstrap host's root context to the top level context where the dump should begin. The utility recursively dumps subcontexts below this point. It defaults to an empty string, that is, the bootstrap host root context.

-format{jndi | ins}

Option	Description
jndi	The default. Displays name components as atomic strings.
ins	Shows name components parsed per INS rules (id.kind).

-report {short | long}

Option	Description
short	The default. Dumps the binding name and bound object type. This output is also provided by <code>JNDI Context.list()</code> .

Option	Description
long	<p>Dumps the binding name, bound object type, local object type, and string representation of the local object (that is, the IORs, string values, and other values that are printed).</p> <p>For objects of user-defined classes to display correctly with the long report option, it may be necessary to add their containing directories to the list of directories searched. Set the environment variable WAS_USER_DIRS. The value can include one or more directories, as for example:</p> <p>Platform</p> <p>Command</p> <p>UNIX WAS_USER_DIRS=/usr/classdir1:/usr/classdir2 export WAS_USER_DIRS</p> <p>Windows NT set WAS_USER_DIRS=c:\classdir1;d:\classdir2</p> <p>All zip, jar, and class files in the specified directories can then be resolved by the class loader when running dumpNameSpace.</p>

-traceString *"some.package.name.to.trace.*=all=enabled"*

Represents the trace string with the same format as that generated by the servers. The output is sent to the file, DumpNameSpaceTrace.out.

Example: Invoking the name space dump utility

It is often helpful to view a dump of the name space to understand why a naming operation is failing. You can invoke the name space dump utility from the command line or from a program. Examples of each option follow.

Invoking name space dump utility from a command line

Invoke the name space dump utility from the command line by entering the following command:

```
dumpNameSpace -host myhost.mycompany.com -port 901
```

OR

```
dumpNameSpace -url corbaloc:iiop:myhost.mycompany.com:901
```

There are several command line options to choose from. For detailed help, enter the following command:

```
dumpNameSpace -help
```

Invoking name space dump utility from a Java program

You can dump name spaces from a program with the `com.ibm.websphere.naming.DumpNameSpace` API. Refer to the WebSphere Application Server API documentation for details on the `DumpNameSpace` program interface

The following example illustrates how to invoke the name space dump utility from a Java program:

```

{
  ...
  import javax.naming.Context;
  import javax.naming.InitialContext;
  import com.ibm.websphere.naming.DumpNameSpace;
  ...
  java.io.PrintStream filePrintStream = ...
  Context ctx = new InitialContext();
  // Starting context for dump
  ctx = (Context) ctx.lookup("cell/nodes/node1/servers/server1");
  DumpNameSpace dumpUtil =
    new DumpNameSpace(filePrintStream, DumpNameSpace.SHORT);
  dumpUtil.generateDump(ctx);
  ...
}

```

Name space dump utility for java:, local: and server name spaces

Sometimes it is helpful to dump the java: name space for a J2EE application. You cannot use the dumpNameSpace command line utility for this purpose because the application's java: name space is accessible only by that J2EE application. From the WebSphere Application Server scripting tool, you can invoke a NameServer MBean to dump the java: name space for any J2EE application running in that same server process.

There is another name space local to server process which you cannot dump with the dumpNameSpace command line utility. This name space has the URL scheme of local: and is used by the container to bind objects locally instead of through the name server. The local: name space contains references to enterprise beans with local interfaces. There is only one local: name space in a server process. You can dump the local: name space by invoking the NameServer MBean associated with that server process.

Name space dump options

Name space dump options are specified in the MBean invocation as a parameter in character string format. The option descriptions follow.

-startAt *some/subcontext/in/the/tree*

Indicates the path from the name space root context to the top level context where the dump should begin. The utility recursively dumps subcontexts below this point. It defaults to an empty string, that is, the root context.

-report {short | long}

Option	Description
short	The default. Dumps the binding name and bound object type. This output is also provided by JNDI Context.list().
long	Dumps the binding name, bound object type, local object type, and string representation of the local object (that is, the IORs, string values, and other values that are printed).

-root {tree | host | legacy | cell | node | server | default}

Specify the root context of where the dump should start. The default value for -root is *cell*. This option is only valid for server name space dumps.

Option	Description
tree	Dump the tree starting at the tree root context.
host	Dump the tree starting at the server host root context (synonymous with "node").
legacy	Dump the tree starting at the legacy root context.
cell	Dump the tree starting at the cell root context. This is the default option.
node	Dump the tree starting at the node root context (synonymous with "host").
server	Dump the tree starting at the server root context. This is -root default.
default	Dump the tree starting at the initial context which JNDI returns by default for that server type.

-format {jndi | ins}

Specify the format to display name component as atomic strings or parsed according to INS rules (id.kind). This option is only valid for server name space dumps.

Option	Description
jndi	Display name components as atomic strings. This is -format default.
ins	Display name components parsed according to INS rules (id.kind).

NameServer MBean invocation

1. Enter the WebSphere Application Server scripting command prompt.
Invoke a method on a NameServer MBean by using the WebSphere Application Server scripting tool. Enter the scripting command prompt by typing the following command:

Platform	Command
UNIX	wsadmin.sh
Windows NT	wsadmin

Use the -help option for help on using the wsadmin command.

2. Select the NameServer MBean instance to invoke.
Execute the following script commands to select the NameServer instance you want to invoke. For example,

```
set mbean [$AdminControl completeObjectName WebSphere:*,type=NameServer,cell=  
cellName,node=nodeName,process=serverName]
```


where *cellName*, *nodeName*, and *serverName* are the names of the cell, node, and server for the MBean you want to invoke. The specified server must be running before you can invoke a method on the MBean.

You can see a list of all NameServer MBeans current running by issuing the following query:

```
$AdminControl queryNames {*:*,type=NameServer}
```

3. Invoke the NameServer MBean.

java: name space

Dump a java: name space by invoking the `dumpJavaNameSpace` method on the NameServer MBean. Since each server application has its own java: name space, the application must be specified on the method invocation. An application is identified by the application name, module name, and component name. The method syntax follows:

```
$AdminControl invoke $mbean dumpJavaNameSpace {{appName}{modName}{compName}{opts}}
```

where *appName* is the application name, *modName* is the module name, and *compName* is the component name of the java: name space you want to dump. The value for *opts* is the list of name space dump options described earlier in this section. The list can be empty.

local: name space

Dump a java: name space by invoking the `dumpLocalNameSpace` method on the NameServer MBean. Since there is only one local: name space in a server process, you have to specify the name space dump options only.

```
$AdminControl invoke $mbean dumpLocalNameSpace {{opts}}
```

where *opts* is the list of name space dump options described earlier in this section. The list can be empty.

Server name space

Dump a server name space by invoking the `dumpServerNameSpace` method on an application server's NameServer MBean. This provides an alternative way to dump the name space on an application server, much like the `dumpNameSpace` command line utility.

```
$AdminControl invoke $mbean dumpServerNameSpace {{opts}}
```

where *opts* is the list of name space dump options described earlier in this section. The list can be empty.

Name space dump output

Name space dump output is sent to the console. It is also written to the file `DumpNameSpace.log`, in the server's log directory.

Example: Invoking the name space dump utility for java: and local: name spaces

It is often helpful to view the dump of a java: or local: name space to understand why a naming operation is failing. The NameServer MBean running in the application's server process can be invoked from the WebSphere Application Server scripting tool to generate a dump of these name spaces. Examples of NameServer MBean calls to generate dumps of java: and local: name spaces follow.

Dumping a java: name space

Assume you want to dump the java: name space of an application component running in server server1 on node node1 of the cell MyCell. The application name is AcctApp in module AcctApp.war, and the component name is Acct Servlet. The following script commands generate a long format dump of the application's java: name space of that application:

```
set mbean [$AdminControl completeObjectName WebSphere:*,type=NameServer,cell=MyCell,node=node1,process=server1]
$AdminControl invoke $mbean dumpJavaNameSpace {{AcctApp}{AcctApp.war}{Acct Servlet}{-report long}}
```

Dumping a local: name space

Assume you want to dump the local: name space for the server server1 on node node1 of cell MyCell. The following script commands will generate a short format dump of that server's local name space:

```
set mbean [$AdminControl completeObjectName WebSphere:type=NameServer,cell=MyCell,node=node1,process=server1]
$AdminControl invoke $mbean dumpLocalNameSpace {{-report short}}
```

Name space dump sample output

Name space dump output looks like the following example, which is the **SHORT** dump format:

```
Getting the initial context
Getting the starting context
```

```
=====
Name Space Dump
  Provider URL: corbaloc:iiop:localhost:9810
  Context factory: com.ibm.websphere.naming.WsnInitialContextFactory
  Requested root context: cell
  Starting context: (top)=outpostNetwork
  Formatting rules: jndi
  Time of dump: Mon Sep 16 18:35:03 CDT 2002
=====

=====
Beginning of Name Space Dump
=====

  1 (top)
  2 (top)/domain                                javax.naming.Context
  2   Linked to context: outpostNetwork
  3 (top)/cells                                javax.naming.Context
  4 (top)/clusters                            javax.naming.Context
  5 (top)/clusters/Cluster1                   javax.naming.Context
  6 (top)/cellname                             java.lang.String
  7 (top)/cell                                 javax.naming.Context
  7   Linked to context: outpostNetwork
  8 (top)/deploymentManager                    javax.naming.Context
  8   Linked to URL: corbaloc::outpost:9809/NameServiceServerRoot
  9 (top)/nodes                                javax.naming.Context
 10 (top)/nodes/will2                          javax.naming.Context
 11 (top)/nodes/will2/persistent               javax.naming.Context
 12 (top)/nodes/will2/persistent/SomeObject   SomeClass
 13 (top)/nodes/will2/nodename                 java.lang.String
 14 (top)/nodes/will2/domain                   javax.naming.Context
 14   Linked to context: outpostNetwork
 15 (top)/nodes/will2/cell                      javax.naming.Context
 15   Linked to context: outpostNetwork
 16 (top)/nodes/will2/servers                  javax.naming.Context
 17 (top)/nodes/will2/servers/server1         javax.naming.Context
 18 (top)/nodes/will2/servers/will2          javax.naming.Context
```

```

19 (top)/nodes/will2/servers/member2          javax.naming.Context
20 (top)/nodes/will2/node                      javax.naming.Context
20   Linked to context: outpostNetwork/nodes/will2
21 (top)/nodes/will2/nodeAgent                javax.naming.Context
22 (top)/nodes/outpost                        javax.naming.Context
23 (top)/nodes/outpost/node                   javax.naming.Context
23   Linked to context: outpostNetwork/nodes/outpost
24 (top)/nodes/outpost/nodeAgent              javax.naming.Context
24   Linked to URL: corbaloc::outpost:2809/NameServiceServerRoot
25 (top)/nodes/outpost/persistent             javax.naming.Context
26 (top)/nodes/outpost/nodename               java.lang.String
27 (top)/nodes/outpost/domain                 javax.naming.Context
27   Linked to context: outpostNetwork
28 (top)/nodes/outpost/servers                javax.naming.Context
29 (top)/nodes/outpost/servers/server1        javax.naming.Context
30 (top)/nodes/outpost/servers/server1/url    javax.naming.Context
31 (top)/nodes/outpost/servers/server1/url/CatalogDAOURL
31                                             java.net.URL
32 (top)/nodes/outpost/servers/server1/mail    javax.naming.Context
33 (top)/nodes/outpost/servers/server1/mail/PlantsByWebSphere
33                                             javax.mail.Session
34 (top)/nodes/outpost/servers/server1/TransactionFactory
34                                             com.ibm.ejs.jts.jts.ControlSet$LocalFactory
35 (top)/nodes/outpost/servers/server1/servername
35                                             java.lang.String
36 (top)/nodes/outpost/servers/server1/WSsamples
36                                             javax.naming.Context
37 (top)/nodes/outpost/servers/server1/WSsamples/TechSampDatasource
37                                             TechSamp
38 (top)/nodes/outpost/servers/server1/thisNode
38                                             javax.naming.Context
38   Linked to context: outpostNetwork/nodes/outpost
39 (top)/nodes/outpost/servers/server1/cell     javax.naming.Context
39   Linked to context: outpostNetwork
40 (top)/nodes/outpost/servers/server1/eis     javax.naming.Context
41 (top)/nodes/outpost/servers/server1/eis/DefaultDatasource_CMP
41                                             Default_CF
42 (top)/nodes/outpost/servers/server1/eis/WSsamples
42                                             javax.naming.Context
43 (top)/nodes/outpost/servers/server1/eis/WSsamples/TechSampDatasource_CMP
43                                             TechSamp_CF
44 (top)/nodes/outpost/servers/server1/eis/jdbc
44                                             javax.naming.Context
45 (top)/nodes/outpost/servers/server1/eis/jdbc/PlantsByWebSphereDataSource_CMP
45                                             PLANTSDB_CF
46 (top)/nodes/outpost/servers/server1/eis/jdbc/petstore
46                                             javax.naming.Context
47 (top)/nodes/outpost/servers/server1/eis/jdbc/petstore/PetStoreDB_CMP
47                                             PetStore_CF
48 (top)/nodes/outpost/servers/server1/eis/jdbc/CatalogDB_CMP
48                                             Catalog_CF
49 (top)/nodes/outpost/servers/server1/jta     javax.naming.Context
50 (top)/nodes/outpost/servers/server1/jta/usertransaction
50                                             java.lang.Object
51 (top)/nodes/outpost/servers/server1/DefaultDatasource
51                                             Default Datasource
52 (top)/nodes/outpost/servers/server1/jdbc     javax.naming.Context
53 (top)/nodes/outpost/servers/server1/jdbc/CatalogDB
53                                             CatalogDB
54 (top)/nodes/outpost/servers/server1/jdbc/petstore
54                                             javax.naming.Context
55 (top)/nodes/outpost/servers/server1/jdbc/petstore/PetStoreDB
55                                             PetStoreDB
56 (top)/nodes/outpost/servers/server1/jdbc/PlantsByWebSphereDataSource
56                                             PLANTSDB
57 (top)/nodes/outpost/servers/outpost         javax.naming.Context
57   Linked to URL: corbaloc::outpost:2809/NameServiceServerRoot
58 (top)/nodes/outpost/servers/member1         javax.naming.Context
59 (top)/nodes/outpost/cell                     javax.naming.Context
59   Linked to context: outpostNetwork
60 (top)/nodes/outpostManager                  javax.naming.Context
61 (top)/nodes/outpostManager/domain           javax.naming.Context
61   Linked to context: outpostNetwork
62 (top)/nodes/outpostManager/cell             javax.naming.Context

```

```

62   Linked to context: outpostNetwork
63 (top)/nodes/outpostManager/servers          javax.naming.Context
64 (top)/nodes/outpostManager/servers/dmgr     javax.naming.Context
64   Linked to URL: corbaloc::outpost:9809/NameServiceServerRoot
65 (top)/nodes/outpostManager/node            javax.naming.Context
65   Linked to context: outpostNetwork/nodes/outpostManager
66 (top)/nodes/outpostManager/nodename        java.lang.String
67 (top)/persistent                           javax.naming.Context
68 (top)/persistent/cell                      javax.naming.Context
68   Linked to context: outpostNetwork
69 (top)/legacyRoot                           javax.naming.Context
69   Linked to context: outpostNetwork/persistent
70 (top)/persistent/AnotherObject             AnotherClass

```

```

=====
End of Name Space Dump
=====

```

Naming and directories: Resources for learning

Use the following links to find relevant supplemental information about naming and directories. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

Programming specifications

- [OMG CosNaming Interoperable Naming Specification](#)

Chapter 14. Managing user profiles

Note: User Profile Manager API is deprecated in the current release, and there is no replacement available.

IBM WebSphere Application Server provides a service for processing user profiles, called the *User Profile Manager*. The service is provided in the form of an EJB entity bean that servlets can call whenever they are required to access a user profile.

The key activities for implementing user profiles are summarized.

1. Customize the user profile support as necessary. Options include:
 - Using the data representation class with the name-value pairs it currently supports (no action required)
 - Extending the data representation class to support additional, arbitrary name-value pairs
 - Adding columns to the base user profile representation
 - Extending the User Profile enterprise bean to import existing databases

Evaluate whether the user profile representation provided by IBM represents the kind of data you want to keep about your users. You might find it desirable to customize the IBM user profile support.

2. Create or modify servlets to use the User Profile Manager and related user profile support classes to maintain user profiles on behalf of Web applications.
3. Assemble your application.
4. Deploy your application.
5. Ensure the administrator appropriately configures User Profile Managers using `userprofile.xml` file. If the programmer and administrator are not the same person, the programmer might need to provide settings information to the administrator, based on how the programmer implemented user profiles.

User profile

Some applications collect data about the users with which they interact. The data is stored in a database. The next time the user interacts with the application, the application recalls the data.

Because the application already knows something about the user, it can provide the user with a more personalized experience.

User profiles provide a means by which a company can maintain and manage database tables containing fields for demographic data and use those tables to interact with a database of individual customers or other users on the company system.

For example, when a repeat user logs onto a Web site that supports user profiles, the Web site can display headlines and advertising tailored to the shopping preferences of that user. The site can address the user by logon name.

An application implementing user profiles requires database access for storing the user profile data it gathers.

UserProfileManager class

Servlets and other application building blocks requiring user profile support should make calls to the class:

```
com.ibm.websphere.userprofile.UserProfileManager
```

The class supports the following functions:

- Creating and deleting user profiles
- Getting and updating (cached and immediate) to and from the database
- Getting user profiles for read-only tasks
- Performing queries on database columns

User profile development options

The application developer has a few options for customizing the user profile support provided by IBM WebSphere Application Server. The Related information provides instructions and additional details about each option.

Extending the data represented in user profiles

Web applications can maintain several pieces of data about users. You can extend the data representation to allow the collection of arbitrary name-value pairs.

Use the following interface with the `com.ibm.websphere.userprofile.UserProfileExtender` class to extend a user profile hash table:

```
com.ibm.websphere.userprofile.UserProfileProperties
```

This action enables you to place arbitrary name-value pairs in the user profile. Extending the hash table is similar to using the `java.util.Dictionary` class in the base JDK 1.x, or any of the classes that extend it.

Adding columns to the base user profile implementation

Application developers can customize user profiles by adding columns to the base user profile implementation. Adding new columns is accomplished by implementing the interface:

```
com.ibm.websphere.userprofile.UserProfileExtender
```

and extending the base class:

```
com.ibm.servlet.personalization.userprofile.UserProfile
```

The application developer can add columns to but not delete columns from the base implementation.

Adding columns is a two-step process, as follows:

1. Extend the `UserProfile` class.
2. Modify your existing servlets to use the new columns.

Several examples are available to demonstrate how to extend the base user profile implementation and utilize the extension with a servlet.

Example	Description
UPServletExample.java	Demonstrates how a servlet opens a user profile and prints the fields contained within.
UserProfileExtendedSample.java	Shows how to extend the UserProfile class to add a column to the user profile for a cellular phone number. The WebSphere Application Server administrator configures the User Profile Manager to point to the extended class.
UPServletExampleExtended.java	Shows how to modify the UPServletExample servlet to include the cellular phone number in the output.
UserProfileExtended.java	Shows how to extend a hash table to place arbitrary name-value pairs into the user profile.
UPServletExtended.java	Shows how to extend the servlet. When any of the newly added columns are removed or replaced, look for the table named "USERPROFILE" in the database to which the user profile is configured and drop that table.

The examples are encoded in HTML for viewing in a browser. The documentation directory also contains nonHTML versions (.java files) that are ready for use.

Extending the User Profile enterprise bean and importing legacy databases

Application developers can extend the User Profile enterprise bean itself and import legacy databases into the user profile. The main advantage in extending the User Profile enterprise bean is to gain the ability to import existing databases into the user profile. You can also extend this enterprise bean to add columns to the base user profile implementation.

Example: UPServletExample.java

```
import java.io.*;
import java.util.*;

import javax.servlet.*;
import javax.servlet.http.*;

import com.ibm.websphere.userprofile.UserProfile;
import com.ibm.websphere.userprofile.UserProfileManager;
import com.ibm.websphere.userprofile.UserProfileCreateException;
import com.ibm.websphere.userprofile.UserProfileFinderException;
import com.ibm.websphere.userprofile.UserProfileRemoveException;

//Creates a Userprofile using the new API

public class UPServlet_ReadWrite extends HttpServlet {

    public void doGet (HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {

        PrintWriter out;
        res.setContentType("text/html");
        out = res.getWriter();
```

```

UserProfileManager manager = UserProfileManager.getUserProfileManager();
UserProfile userprofile;

try {
    //Try creating the UserProfile
    userprofile = manager.addUserProfile("bpink");
} catch(UserProfileCreateException e1) {
    try {
        //Try finding the existing in readWrite mode.
        //Second argument indicates whether we want to get userprofile
        //in read only mode or read write mode.

        userprofile = manager.getUserProfile("bpink",true);
    } catch(UserProfileFinderException e) {
        e.printStackTrace();
        return;
    }
}

//Set the properties

userprofile.setAddress1("myaddress1");
userprofile.setAddress2("myaddress2");
userprofile.setFirstName("Pinkowski");
userprofile.setSurName("Ben");
userprofile.setDayPhone("555-6677");
userprofile.setNightPhone("556-6765");
userprofile.setCity("MYCITY");
userprofile.setNation("myCountry");
userprofile.setEmployer("MyEmployer");
userprofile.setFax("7823470");

userprofile.setLanguage("mylanguage");
userprofile.setEmail("MyEmail@email");
userprofile.setStateOrProvince("myState");
userprofile.setPostalCode("xxxxx");

//Freeing resources held by userprofile
manager.releaseResources(userprofile);
userprofile=null;

//Checking whether it updated the info

try {
    //Getting the existing userprofile in ReadOnly mode.

    userprofile = manager.getUserProfile("bpink",false);
} catch(UserProfileFinderException e1) {

    out.println("Error finding ");
    e1.printStackTrace();
    return;
}

```



```

        //Displaying the properties of userprofile

        out.println(userprofile.getAddress1()+"<br>");
        out.println(userprofile.getAddress2()+"<br>");;
        out.println(userprofile.getFirstName()+"<br>");;
        out.println(userprofile.getSurName()+"<br>");;
        out.println(userprofile.getDayPhone()+"<br>");;
        out.println(userprofile.getNightPhone()+"<br>");;
        out.println(userprofile.getCity()+"<br>");;
        out.println(userprofile.getNation()+"<br>");;
        out.println(userprofile.getEmployer()+"<br>");;
        out.println(userprofile.getFax()+"<br>");;
        out.println(userprofile.getLanguage()+"<br>");;
        out.println(userprofile.getEmail()+"<br>");;
        out.println(userprofile.getStateOrProvince()+"<br>");;
        out.println(userprofile.getPostalCode()+"<br>");;

        //Freeing resources held by userprofile
        manager.releaseResources(userprofile);
    }
}

```

Example: UserProfileExtendedSample.java

```

/* -----
** Copyright 1997-99 IBM Corporation. All rights reserved.
**
** -----
*/
package com.ibm.servlet.personalization.userprofile;

import com.ibm.servlet.personalization.userprofile.UserProfile;
import com.ibm.websphere.userprofile.UserProfileExtender;

//Extensions of UserProfile to add new Columns should implement UserProfileExtender
public class UserProfileExtendedSample
    extends com.ibm.servlet.personalization.userprofile.UserProfile
    implements UserProfileExtender {

    //New column that is being added by this
    //derived class.
    public String cellPhone;

    //Manager Class will call this method to append new Column types.
    //If UserProfile class is extended to append new columns
    //TOTAL COLUMNS: Base Class columns + columns returned by this class

    public String[] getNewColumns() {
        //If variable name is "cellPhone," you need to
        //return "cellPhone" in array format. JDBC equivalent will be
        //generated automatically. You can add multiple columns.
        //For multiple columns: String newCol={"fieldName1","fieldName2",...};
        String[] newCol={"cellPhone"};
        return newCol;
    }

    public String getCellPhone() {
        // Need to call this method to
        // get the things from persistence store.
        return(String)getByType("cellPhone");
    }

    public void setCellPhone(String value) {
        cellPhone = value;
    }
}

```

```

        //Call this method to store the
        //things in persistence store
        setByType("cellPhone", value);
    }
}

```

Example: UPServletExampleExtended.java

```

import java.io.*;
import java.util.*;

import javax.servlet.*;
import javax.servlet.http.*;

import com.ibm.websphere.userprofile.UserProfile;
import com.ibm.websphere.userprofile.UserProfileManager;
import com.ibm.websphere.userprofile.UserProfileCreateException;
import com.ibm.websphere.userprofile.UserProfileFinderException;
import com.ibm.websphere.userprofile.UserProfileRemoveException;
import com.ibm.servlet.personalization.userprofile.UserProfileExtendedSample;

public class UPServletExtendedSample extends HttpServlet {

    public void doGet (HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
    {
        UserProfileManager manager = UserProfileManager.getUserProfileManager();
        UserProfile userprofile;

        PrintWriter out;

        res.setContentType("text/html");
        out = res.getWriter();

        try {

            //try Creating the UserProfile

            userprofile = manager.addUserProfile("bpink");

        } catch( UserProfileCreateException e1) {

            try { //try finding the existing in readWrite mode

                userprofile = manager.getUserProfile("bpink",true);
            } catch(UserProfileFinderException e) {
                e.printStackTrace();
                return;
            }

        }

        userprofile.setAddress1("myaddress1");
        userprofile.setAddress2("myaddress2");
        userprofile.setFirstName("Pinkowski");
        userprofile.setSurName("Ben");
        userprofile.setDayPhone("555-6677");
        userprofile.setNightPhone("556-6765");
        userprofile.setCity("MYCITY");
        userprofile.setNation("myCountry");
        userprofile.setEmployer("MyEmployer");
        userprofile.setFax("7823470");
    }
}

```

```

userprofile.setLanguage("mylanguage");
userprofile.setEmail("MyEmail@email");
userprofile.setStateOrProvince("myState");
userprofile.setPostalCode("xxxxx");

//calling setCellPhone
((com.ibm.servlet.personalization.userprofile.UserProfileExtendedSample)
    userprofile).setCellPhone("346-4588");

//Freeing resources held by userprofile
manager.releaseResources(userprofile);
userprofile=null;

//Checking whether it updated the info
try {
    //Getting the existing userprofile
    userprofile = manager.getUserProfile("bpink",false);

} catch( UserProfileFinderException e1) {
    out.println("Error finding ");
    e1.printStackTrace();
    return;
}

out.println(userprofile.getAddress1()+"<br>");
out.println(userprofile.getAddress2()+"<br>");
out.println(userprofile.getFirstName()+"<br>");
out.println(userprofile.getSurName()+"<br>");
out.println(userprofile.getDayPhone()+"<br>");
out.println(userprofile.getNightPhone()+"<br>");
out.println(userprofile.getCity()+"<br>");
out.println(userprofile.getNation()+"<br>");
out.println(userprofile.getEmployer()+"<br>");
out.println(userprofile.getFax()+"<br>");
out.println(userprofile.getLanguage()+"<br>");
out.println(userprofile.getEmail()+"<br>");
out.println(userprofile.getStateOrProvince()+"<br>");
out.println(userprofile.getPostalCode()+"<br>");

//Calling getCellPhone
out.println(((UserProfileExtendedSample)userprofile).getCellPhone()+"<br>");

//Freeing resources held by userprofile
manager.releaseResources(userprofile);
userprofile=null;

//For getting values by cellPhone
out.println("<br><br>Retreiving by Cell Phone <br>");
Enumeration enum = manager.findUserProfiles("cellPhone","346-4588");
while(enum.hasMoreElements()) {
    com.ibm.websphere.userprofile.UserProfile up =
        (com.ibm.websphere.userprofile.UserProfile)enum.nextElement();
    out.println("first name :"+up.getFirstName()+"<br>");

    //Freeing resources held by userprofile

```

```

        manager.releaseResources(up);
    }
}

```

Example: UserProfileExtended.java

```

package com.ibm.servlet.personalization.userprofile;
/* -----
** Copyright 1997-99 IBM Corporation. All rights reserved.
** -----
*/
import java.util.*;

import com.ibm.servlet.personalization.userprofile.UserProfile;

import com.ibm.websphere.userprofile.UserProfileExtender;
import com.ibm.websphere.userprofile.UserProfileProperties;

public class UserProfileExtended extends UserProfile implements UserProfileExtender,
    UserProfileProperties {
    //New column that is being added by this
    //derived class.
    public Hashtable properties;

    static String propCol ="properties";

    //Manager Class will call this method to append new Column types
    //to SQL Strings. If UserProfile class is extended to append new columns
    //it should implement UserProfileExtender.
    //COLUMNS: Base Class columns + columns returned by this class

    public String[] getNewColumns() {
        //if variable name is properties, you need to
        //return "properties" . JDBC equivalent will be
        //generated automatically.
        String[] newCol={propCol};
        return newCol;
    }

    public Object getValue(String key) {
        // Need to call this method to
        // get the things from persistent store
        properties = (Hashtable) getByType(propCol);

        if(properties != null)
            return properties.get(key);

        else return null;
    }

    public void putValue(String key, Object value) {
        properties =(Hashtable) getByType(propCol);

        if(properties == null)
            properties = new Hashtable();

        properties.put(key,value);

        //store in persistent store
        setByType(propCol, properties);
    }

    public void removeValue(String key) {

```

```

        properties = (Hashtable) getByType(propCol);
        if(properties == null)
            return;

        properties.remove(key);

        //store in persistent store
        setByType(propCol, properties);
    }
}

```

Example: UPServletExtended.java

```

import java.io.*;
import java.util.*;

import javax.servlet.*;
import javax.servlet.http.*;

import com.ibm.websphere.userprofile.UserProfile;
import com.ibm.websphere.userprofile.UserProfileManager;
import com.ibm.websphere.userprofile.UserProfileCreateException;
import com.ibm.websphere.userprofile.UserProfileFinderException;
import com.ibm.websphere.userprofile.UserProfileRemoveException;
import com.ibm.websphere.userprofile.UserProfileProperties;

public class UPServletExtended extends HttpServlet {

    public void doGet (HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        UserProfileManager manager = UserProfileManager.getUserProfileManager();
        UserProfile userprofile;

        PrintWriter out;

        res.setContentType("text/html");
        out = res.getWriter();

        try {

            //try Creating the UserProfile

            userprofile = manager.addUserProfile("bpink");

        } catch( UserProfileCreateException e1) {

            try { //try finding the existing in readWrite mode

                userprofile = manager.getUserProfile("bpink",true);
            } catch(UserProfileFinderException e) {
                e.printStackTrace();
                return;
            }

        }

        userprofile.setAddress1("myaddress1");
        userprofile.setAddress2("myaddress2");
        userprofile.setFirstName("Pinkowski");
        userprofile.setSurName("Ben");
        userprofile.setDayPhone("555-6677");
        userprofile.setNightPhone("556-6765");
        userprofile.setCity("MYCITY");
    }
}

```

```

userprofile.setNation("myCountry");
userprofile.setEmployer("MyEmployer");
userprofile.setFax("7823470");
userprofile.setLanguage("mylanguage");
userprofile.setEmail("MyEmail@email");
userprofile.setStateOrProvince("myState");
userprofile.setPostalCode("xxxxx");

//calling putValue

((UserProfileProperties)userprofile).putValue("name", "HHHHHHH");
((UserProfileProperties)userprofile).putValue("Date", new java.util.Date());

//Freeing resources held by userprofile
manager.releaseResources(userprofile);
userprofile=null;

//Checking whether it updated the info
try {

    //Getting the existing userprofile

    userprofile = manager.getUserProfile("bpink", false);

    out.println(userprofile.getAddress1()+"<br>");
    out.println(userprofile.getAddress2()+"<br>");
    out.println(userprofile.getFirstName()+"<br>");
    out.println(userprofile.getSurName()+"<br>");
    out.println(userprofile.getDayPhone()+"<br>");
    out.println(userprofile.getNightPhone()+"<br>");
    out.println(userprofile.getCity()+"<br>");
    out.println(userprofile.getNation()+"<br>");
    out.println(userprofile.getEmployer()+"<br>");
    out.println(userprofile.getFax()+"<br>");
    out.println(userprofile.getLanguage()+"<br>");
    out.println(userprofile.getEmail()+"<br>");
    out.println(userprofile.getStateOrProvince()+"<br>");
    out.println(userprofile.getPostalCode()+"<br>");

    //Getting the values

    out.println(((UserProfileProperties)userprofile).getValue("name")+"<br>");
    out.println(((UserProfileProperties)userprofile).getValue("Date")+"<br>");
    out.println("Removing Values ");
    ((UserProfileProperties)userprofile).removeValue("name");
    ((UserProfileProperties)userprofile).removeValue("Date");
    out.println(((UserProfileProperties)userprofile).getValue("name")+"<br>");
    out.println(((UserProfileProperties)userprofile).getValue("Date")+"<br>");

    //Freeing resources held by userprofile
    manager.releaseResources(userprofile);

} catch( UserProfileFinderException e1) {

    out.println("Error finding ");
    e1.printStackTrace();
    return;
}
}
}

```

userprofile.xml

To *installation_root*/properties, add a file named userprofile.xml in the following format. Specify enterprise bean class names; data wrapper class name; and JNDI names for the read-only bean, read/write bean, and data source (from step 2). You must also add user ID and password information for the JNDI data source

The following example file contains class names as provided in WebSphere Application Server. If data wrapper and enterprise bean class names are extended programmatically, change them accordingly.

```
<?xml version="1.0"?>

  <userprofile>
    <userprofile-enabled>true</userprofile-enabled>
    <userprofile-wrapper-class>
      <classname>
        com.ibm.servlet.personalization.userprofile.UserProfile
      </classname>
    </userprofile-wrapper-class>
    <userprofile-manager-name>
      User Profile Manager
    </userprofile-manager-name>
    <userprofile-bean>
      <readonly-interface>
        com.ibm.servlet.personalization.userprofile.UP_ReadOnly
      </readonly-interface>
      <readwrite-interface>
        com.ibm.servlet.personalization.userprofile.UP_ReadWrite
      </readwrite-interface>
      <readonlyhome-interface>
        com.ibm.servlet.personalization.userprofile.UP_ReadOnlyHome
      </readonlyhome-interface>
      <readwritehome-interface>
        com.ibm.servlet.personalization.userprofile.UP_ReadWriteHome
      </readwritehome-interface>
      <readonly-JNDI-lookupName>UP_ReadOnlyHome</readonly-JNDI-lookupName>
      <readwrite-JNDI-lookupName>UP_ReadWriteHome</readwrite-JNDI-lookupName>
    </userprofile-bean>

    <userprofile-store>
      <database-userid></database-userid>
      <database-password></database-password>
      <database-datasource></database-datasource>
    </userprofile-store>
  </userprofile>
```

Chapter 15. Assembling applications

Assemble application modules (known as EAR files) from new or existing J2EE 1.3 modules, including these archives: Web application archives (WAR), resource adapter archives (RAR), enterprise beans (EJB JAR), and application client archives (JAR). This packaging and configuration of code artifacts into application modules or standalone Web modules is necessary for deploying the applications onto the application server.

Before you begin

Gather the code artifacts that you want to package into one or more assembled modules. Code artifacts include these items that you have created and unit tested in your favorite integrated development environment:

- Enterprise beans
- Servlets, JavaServer Pages (JSP) files and other Web components
- Resource adapter (*connector*) implementations
- Application clients
- Other supporting classes and files

Steps for this task

1. Start the Application Assembly Tool (AAT).
2. **(Optional)** Migrate existing J2EE 1.2 modules to J2EE 1.3.

The Application Assembly Tool (AAT) has an option for migrating J2EE 1.2 application modules to J2EE 1.3. The J2EE 1.2 module is kept intact, with a new 1.3 module created. See also the `earconvert` tool documentation.

You must migrate J2EE 1.2 application modules to which you want to add J2EE 1.3 level WAR, RAR, EJB and client modules. This tool migrates only the application modules. J2EE 1.2-level modules inside a J2EE 1.2 application module must be migrated by other means.

3. Assemble new EJB modules (enterprise bean JAR files) as needed.

Assemble an EJB module to contain enterprise beans and related code artifacts. (Group Web components, client code, and resource adapter code in separate modules.)

You can install an EJB module as a standalone application or you can combine it with other modules into an enterprise application.

4. Assemble new Web modules (WAR files) as needed.

Assemble a Web module to contain servlets, JSP files, and related code artifacts. (Group enterprise beans, client code, and resource adapter code in separate modules.)

You can install a Web module as a standalone application or combine it with other modules into an enterprise application.

5. Assemble new application client modules (client JAR files) as needed.
6. Assemble new resource adapter archives (RAR files) as needed.

Assemble a resource adapter archive module to contain the library implementation code that your application uses to connect to enterprise information systems (EIS). (Group enterprise beans, Web components, and client code in separate modules.)

7. Assemble an application module from other module types.
You are ready to combine your new or migrated modules into an application module (EAR file).
For applications containing only Web modules, this step is optional. It is feasible to deploy Web modules without assembling them into application modules.
8. Verify your archive files.
Verify your archive files and correct any problems so that generation of deployment code is successful. During verification, the Application Assembly Tool (AAT) checks that an archive file is complete, and that deployment descriptor properties and references contain appropriate values.
9. Remember to save your application one last time.
10. Generate code for deployment for applications containing EJB modules.
If the application modules contain EJB modules, you must generate deployment code for the enterprise beans in the application before you deploy applications on the server. The Application Assembly Tool (AAT) provides this ability, or you can use the `ejbdeploy` command line tool.
11. Open existing modules (**File > Open**) in the AAT to modify them as needed.
For example, you can add or remove modules and edit deployment descriptor properties.

What to do next

After assembling your applications, use a systems management tool to deploy the EAR or WAR files onto the application server.

The systems management tool follows the security and deployment instructions defined in the deployment descriptor, and enables you to modify bindings specified within the AAT. The tool locates the required external resources that the application uses, such as enterprise beans and databases.

Select a tool to use:

- Deploying and managing applications with the GUI
- Deploying and managing applications using programming (Java administrative APIs)
- Deploying and managing applications using scripting (WebSphere Application Server `wsadmin` tool)

If you are uncertain of which systems management tool to use, try using the administrative console.

Application assembly and J2EE applications

Application assembly is the process of creating an Enterprise Archive (EAR) file containing all files related to an application, as well as an XML deployment descriptor for the application. This configuration and packaging prepares the application for deployment onto an application server.

EAR files are comprised of the following archives:

- Enterprise bean (JAR) files (known as EJB modules)
- Web application (WAR) files (known as Web modules)
- Application client (JAR) files (known as client modules)

- Resource adapter (RAR) files (known as resource adapter modules)

Ensure that modules are contained in an EAR file so that they may be deployed onto the server. The exceptions are WAR modules, which you can deploy individually. Although WAR modules can contain regular JAR files, they cannot contain the other module types described previously.

The assembly process includes the following:

- Selecting all of the files to include in the module
- Creating a deployment descriptor containing instructions for module deployment on the application server.

As you configure properties using the Application Assembly Tool (AAT), the tool generates the deployment descriptor for you. While the AAT graphical interface is recommended, you can also edit descriptors directly (`../javadoc/wccm/index.html`) in your favorite XML editor.

- Packaging modules into a single Enterprise archive (EAR) file, which contains one or more files in a compressed format

Archive support in Version 5.0

These archives and Web components are supported:

- J2EE 1.3 Enterprise application (EAR) files
- EJB 2.0 (JAR) files
- Servlet 2.3 Web application WAR files
- Application Client 1.3 JAR files
- Connector 1.0 RAR files

These archive files and Web components are back-level and may be read but not created or changed:

- J2EE 1.2 EAR files
- EJB 1.1 JAR files
- Servlet 2.2 WAR files
- Application Client 1.2 JAR files

Starting the Application Assembly Tool (AAT)

Before you begin

A graphical interface is available for packaging code artifacts into various archives (modules) and configuring their J2EE 1.3 compliant deployment descriptors. The Application Assembly Tool (AAT) is available from the Windows Start menu, or you can invoke the tool from a command line as described in the Steps for this task.

If you access the Application Assembly Tool from a remote browser and select the Help, the Help files do not display. You can only view the Help files from a locally installed browser. To view the Help files and avoid this problem, close all the Netscape sessions on the remote machine and click **Help**. A new Netscape session starts, and you can then view the Help files.

Steps for this task

1. Change directory at a system command prompt to the location of the assembly.bat|sh file, typically install_root/bin.
2. Run the assembly script to launch the graphical interface.
3. Select whether to work with an existing module or create a new one.

Results

The navigation tree displays a hierarchical structure used to build the contents of a new module, or to work with the contents of an existing module. Icons in the tree represent the components, assembly properties, and files for the module. The assembly properties appear in the AAT workspace.

Starting the Application Assembly Tool on a UNIX platform results in errors similar to the following:

```
...Font specified in font.properties not found [-urw-itc
zapfdingbats-medium-r-normal--*%d--*p-*sun-fontspecific]
Font specified in font.properties not found [-urw-itc
zapfdingbats-medium-r-normal--*%d--*p-*sun-fontspecific]
...
```

The Application Assembly Tool or installer functions are not affected by these errors. These messages display in the command shell that spawned the Java GUI. You can disregard these messages.

What to do next

Consider whether you have any existing J2EE 1.2 application modules that you would like to migrate to J2EE 1.3.

You can create new modules of the following types, to assemble into an application module later:

- Assembling EJB modules
- Assembling Web modules
- Assembling application client modules
- Assembling resource adapter modules

Rather than create new modules to assemble an application, you can proceed directly to assembling a new application module. While assembling an application module, you can create any new modules that you need.

Migrating application modules from J2EE 1.2 to J2EE 1.3

The Application Assembly Tool (AAT) has an option for migrating J2EE 1.2 application modules to J2EE 1.3. The J2EE 1.2 module is kept intact, with a new 1.3 module created. See also the earconvert tool documentation.

Migrate J2EE 1.2 application modules to which you want to add J2EE 1.3 level Web application (WAR) modules, Resource adapter (RAR) modules, Entity bean (EJB) modules, and application client modules. This tool migrates only the application modules. Migrate J2EE 1.2-level modules inside a J2EE 1.2 application module by other means.

Note: When Entity beans are moved from a J2EE 1.2 module to a J2EE 1.3 module, the EJB container will then apply rules defined in the EJB 2.0 specification to these

beans. The EJB 2.0 specification mandates that when a `findBy` method is called on a bean home (except for `findByPrimaryKey`), the EJB container must cause other Entity beans enlisted in the same transaction to write out their current state to the persistent store. This is to ensure that the `findBy` operation is performed on the most current data. Application developers should plan for and be aware of any changes to the application behavior as a result of this rule.

Steps for this task

1. Start the AAT.
2. Use it to open the J2EE 1.2 application module you want to migrate.
3. Click **Convert EAR** from the file menu.
4. Save the new J2EE 1.3 application.

What to do next

Assemble zero or more new modules of your choice:

- Assembling EJB modules
- Assembling Web modules
- Assembling application client modules
- Assembling resource adapter modules

Another option is to proceed directly to assembling a new application module. You can create any new modules that you need, while assembling an application module.

earconvert tool

A command line tool is provided for migrating J2EE 1.2 application modules to J2EE 1.3. This migration enables you to add J2EE 1.3 modules to the migrated application module. See also the Application Assembly Tool (AAT) for information on performing this task.

Migrate J2EE 1.2 application modules to which you want to add J2EE 1.3 level Web application (WAR), Resource adapter (RAR), Enterprise beans (EJB), and client modules. This tool migrates only the application modules. Ensure that you migrate J2EE 1.2-level modules inside a J2EE 1.2 application to prevent working with back-level files.

Syntax

Locate the file that contains the `earconvert.bat` | `sh` tool, which must reside in the `bin` directory of the product installation root.

earconvert

j2ee_1.2_file_name

j2ee_1.3_file_name

Parameters

Supported arguments include:

"j2ee_1.2_file_name"

Specifies the actual name of the existing J2EE 1.2 application file. (In this and other arguments, use quotation marks to allow for path names that contain spaces.)

"j2ee_1.3_file_name"

Specifies what you would like to name the new J2EE 1.3 application file.

Examples

The following command creates a new J2EE 1.3 archive, `new_application.ear`, based on the J2EE 1.2 archive, `existing_application.ear`.

```
earconvert existing_application.ear new_application.ear
```

Assembling new or modifying existing modules

Before you begin

Ensure that code artifacts, such as servlets, JSP files, enterprise beans, and application clients are assembled into their respective modules.

If you want to use existing J2EE 1.2 modules in your J2EE 1.3 application, migrate these modules to J2EE 1.3 first. Also migrate any J2EE 1.2 application modules to which you want to add J2EE 1.3 modules.

You are now ready to combine your new or migrated modules into an application module Enterprise application (EAR file).

The Application Assembly Tool (AAT) provides flexibility in assembling applications from various Web application (WAR), Resource adapter (RAR), Enterprise beans (EJB JAR), and application client (JAR) files. Options described in assembling applications include:

- Importing an existing module (JAR, RAR or WAR file)
- Creating a new module while you create the new application
- Copying code artifacts, such as servlets, from one module to another of the same type, to reside in the new application

Steps for this task

1. Start the AAT.
2. From the New tab, select **Application**, and click **OK**, if you did not already specify to create a new application module.
Each of the next three steps is optional, but you must perform at least one of them.
3. **(Optional)** Import existing modules into the application module.
 - a. Right-click the folder for the type of module you want to import, such as an EJB module, in the navigation tree.
 - b. Click **Import** from its right-click menu.
 - c. Use the file browser to locate and select the archive file for the module.
 - d. Click **Open**. The archive file appears under the appropriate folder in the navigation tree.
 - e. Click the plus sign (+) next to the icon for the archive, to view the module contents and edit its properties if needed.
 - f. Save the application module.
4. **(Optional)** Create a new archive file to include in the application.

- a. Right-click the folder for the type of module to create (such as enterprise beans (EJB) modules, Web application modules (WAR), resource adapter (RAR) files, or application client modules) in the navigation tree.
 - b. Click **New** from its right-click menu.
 - c. Configure properties of the new module when it displays.
 - d. Click **OK**. The archive file displays under the appropriate folder.
 - e. Click the plus sign (+) to verify file contents and enter assembly properties.
 - f. Add enterprise beans, if this is an EJB module.
 - g. Right-click the folder corresponding to the type of bean to create (session bean or entity bean), and click **New** or **Import**.
 - h. Configure properties of the enterprise bean when it displays.
 - i. Click **OK**. The enterprise bean appears in the navigation pane.
 - j. Click the plus sign (+) to verify file contents and enter assembly properties.
 - k. Save the application module.
5. **(Optional)** Copy code artifacts, such as servlets, from one module to another of the same type, to reside in the new application.
 - a. Identify the code artifact to copy, and the type of module in which it resides. Make sure you already have the same kind of module (such as a Web module) created in the new application module.
 - b. Open a separate, existing module in the AAT by selecting **File > Open** from the menu bar.
 - c. Arrange the AAT workspace so that you can see both the new application module and the source archive containing the code artifact.
 - d. Copy and paste the code artifact from the source module to the same module type in the new application.
 For example, copy a container-managed persistence (CMP) bean from the source EJB module into the new EJB application module.
 - e. Save the application module.
 6. Continue to add desired modules to the application module.
 7. Define security properties for the application.
 - a. Right-click the **Security Roles** icon in the navigation tree.
 - b. Click **New**.
 - c. Configure the security properties.
 - d. Click **OK**.
 8. Add supplementary files needed by the application.
 - a. Right-click the **Files** icon in the navigation tree, and select **Add Files**.
 - b. Add files, using the Add Files dialog.
 9. Save the application module.

Results

You are performing application assembly results in a J2EE 1.3 compliant EAR file containing one or more WAR, RAR, or JAR files.

Note: If you use the Application Assembly Tool to create application client modules, you must also use the Application Client Resource Configuration Tool. Using this tool, you can define references to resources (other than enterprise beans) on the machine where the application client resides.

What to do next

"Verifying archive files". "Generating code for deployment".

After an application is assembled and you generate your application code for deployment, use the administrative console to install the application in an application server.

Use the administrative console at installation time to carry out the security instructions defined in the deployment descriptor and to locate required external resources, such as enterprise beans and databases. You can add configuration properties and redefine binding properties defined in the Application Assembly Tool.

After the application deploys, you can use the Application Assembly Tool to modify the application by adding or removing modules, editing deployment descriptor properties and regenerating code for deployment.

Note: If your application has a large number of modules, it might not install successfully onto a server. Package your application so the .ear file has as few modules as are necessary. Modules can include metadata for the modules such as information on deployment descriptors, bindings and IBM extensions.

Adding files to assembled modules

Before you begin

Review the usage scenario (as follows) to become familiar with the **Add Files** dialog.

Use the **Add Files** dialog box of the Application Assembly Tool (AAT) to import files into assembled modules including Enterprise application (EAR), Web application (WAR), Resource adapter (RAR) and Application client (JAR) files.

This task assumes that you are performing another task, such as assembling a Web or EJB module, when the **Add Files** dialog is presented to you.

Steps for this task

1. Click **Browse**. Locate the files to add.
 - To add specific individual files, select the directory or archive (WAR, JAR, RAR, ZIP, for example) containing the files.
 - To add an entire directory of files, select its parent directory.
2. Click **OK**.

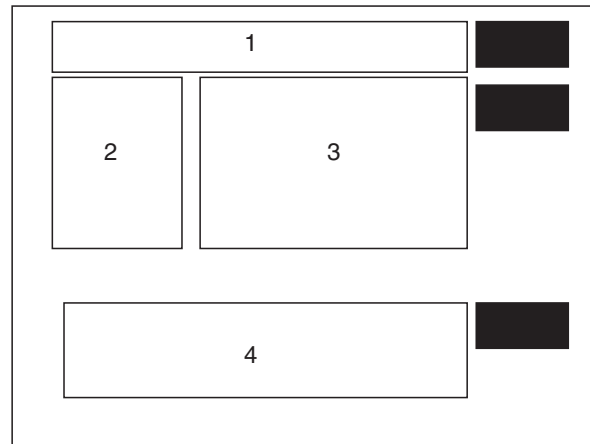
The selected directory or archive appears in the top left part of the dialog box, in an expandable tree.

The top right part of the dialog box shows the contents of the directory, subdirectory, or archive that is selected on the left-hand side.
3. Select one or more items to add from the top right part of the dialog, then click **Add**.

As you add files, they will be displayed in the lower half of the dialog box.
4. Change your left-hand selection to gain access to other files that you want to add, as needed.
5. Click **OK** when all of the files that you want to add appear in the lower half of the dialog box.

Usage scenario

The following example refers to the main areas of the Add Files dialog box. Details such as clicking OK are omitted. Refer to the detailed task steps above for this information.



Suppose you are constructing a new application module and want to add `myFile.txt` file to the archive as a supplementary file. The `myFile.txt` currently is contained within the `myFiles` subdirectory of a JAR file that resides somewhere on your directory system.

1. Browse for the JAR file.
2. Select the JAR file.
3. Exit the browse dialog. At this point:
 - Area 1 of the **Add Files** dialog contains the path to the JAR file.
 - Area 2 displays the JAR file name as the root directory of an expandable tree showing the directories in the JAR file — including `myFiles`.
 - Area 3 shows the root contents of the JAR file, as well as any subdirectories visible from the root.
4. Select the `myFiles` directory from area 2, causing `myFile.txt` to become visible in area 3.
5. Click **myFiles.txt** from area 3 and specify to **Add** the file. Now this file is listed in area 4, the lower half of the dialog, which indicates it is the file that you want to add to the new application module.
6. Exit the **Add Files** dialog.

Resource environment reference assembly settings

Resource environment reference elements contain declarations of an enterprise bean's reference to an administered object associated with a resource in the enterprise bean's environment.

Name

Specifies the name of the resource environment reference.

Its value is the environment entry name used in the enterprise bean code.

Data type

String

Description

Contains the information that the EJB jar file producer wants to provide to the EJB jar file consumer.

Data type String

Type

Specifies the type of a resource environment reference.

Data type String

Resource Adapter Archive file assembly settings

Use this page to set the resource adapter archive file properties.

File name

Specifies the file name of the Resource Adapter Archive.

Data type String

Display name

Specifies a short name that is intended to be displayed by the GUI.

Data type String

Description

Specifies a description that should include any information that the component file producer wants to provide to the consumer of the component file (that is, to the deployer).

Data type String

EIS type

This helps in identifying EIS instances that can be used with this resource adapter.

Data type String

Vendor name

Specifies a string-based version of the resource adapter from the resource adapter provider.

Data type String

Version

Specifies a string-based version of the resource adapter from the resource adapter provider.

Data type String

Specification

Specifies the version of the connector architecture specification that is supported by this resource adapter.

Data type String

License required

Specifies if a license is or is not required.

Description

If a license is required, this field specifies the licensing requirements for the resource adapter module. For example, duration of license, number of connection restrictions, and so forth.

Data type String

Implementation

The element (managedconnectionfactory class) that specifies the fully qualified name of the Java class that implements the javax.resource.spi.ManagedConnectionFactory interface.

Data type Class

Interface

The element (credential-interface) that specifies the interface that the resource adapter implementation supports for the representation of the credentials.

The possible values are:

```
<credential-interface> javax.resource.spi.security.PasswordCredential
</credential-interface>
<credential-interface> javax.resource.spi.security.GenericCredential
</credential-interface>
```

Data type Class

Implementation

The element (connectionfactory class) that specifies the fully-qualified name of the ConnectionFactory class that implements the resource adapter specific ConnectionFactory interface.

Data type Class

Interface

The element (connection-interface) that specifies the fully-qualified name of the Connection interface supported by the resource adapter.

Data type Class

Implementation

The element (connection class) that specifies the fully-qualified name of the Connection class that implements the resource adapter specific Connection interface.

Data type Class

Support Reauthentication

Specifies whether the resource adapter implementation supports re-authentication of existing ManagedConnection instances.

The values are either True or False.

Data type String

Transaction

Specifies the level of transaction support provided by the resource adapter.

The three possible values are:

- NoTransaction
- LocalTransaction
- XATransaction

Data type String

Small Icon

The image is used as an icon to represent the module in a GUI.

Specifies a JPEG or GIF file containing a small image (16x16 pixels).

Data type Image

Large Icon

The image is used as an icon to represent the module in a GUI.

Specifies a JPEG or GIF file containing a small image (32x32 pixels).

Data type Image

Basic Password

The basic user password authentication mechanism that is specific to an EIS.

Credential Interface

Specifies the interface that the resource adapter implementation supports for the representation of the credentials. For Basic Password the credential value is *javax.resource.spi.security.PasswordCredential*.

Description

Any information that describes Basic Password selection.

Data type String

Kerberos V5

Specifies a Kerberos version 5 authentication mechanism.

Credential Interface

Specifies the interface that the resource adapter implementation supports for the representation of the credentials. For Kerberos version 5, the credential value is *javax.resource.spi.security.GenericCredential*.

Description

Any information that describes the Kerberos V5 selection.

Data type String

Property Name

Specifies the name of a configuration property.

The possible values are:

```
<config-property-name>ServerName</config-property-name>  
<config-property-name>PortNumber</config-property-name>  
<config-property-name>UserName</config-property-name>  
<config-property-name>Password</config-property-name>  
<config-property-name>ConnectionURL</config-property-name>
```

Data type String

Property Type

Contains the fully-qualified Java type of a configuration property as required by the ManagedConnectionFactory instance.

Data type String

Property Value

Contains the value of a configuration entry.

Data type String

Description

Describes the parent element.

Data type String

Permission Specification

Specifies a security permission that is required by the resource adapter code.

Data type String

Saving applications after assembly

Periodically save modules that you assemble with the Application Assembly Tool (AAT). Save any changes right before you close the module with which you are working.

Before you begin

This task assumes you have started the AAT and are working with a particular module.

Steps for this task

1. Save the archive file by clicking **File > Save As**.
 - If you are saving an existing archive file or application, click **File > Save**.
2. Name the new archive file or application whatever you like.

This step is optional if you are working with an existing archive file or application.

What to do next

Now that you have saved your assembled application, you can verify your archives and generate code for deployment.

Verifying archive files

Verify your archive files and correct any problems so that generation of deployment code is successful. During verification, the Application Assembly Tool (AAT) checks that an archive file is complete, and that deployment descriptor properties and references contain appropriate values.

This task assumes you have previously assembled and saved one or more modules.

Steps for this task

1. Start the Application Assembly Tool (AAT).
2. Click **File > Open** and select the module to verify.
3. Right-click the name of the module at the top of the navigation pane and click **Verify**.
4. Click **Verify** in the **Verify** window. The tool displays a scrolling window for viewing status messages as the verification proceeds.
5. Save the application.

Results

Archive files have been verified. The following list includes, but is not limited to, areas that the verification process has checked:

- Required deployment properties contain values.
- Values specified for environment entries match their associated Java types.
- In both Enterprise application (EAR) and Web application (WAR) files:
 - The target enterprise bean of the link exists for EJB references.
 - The target role exists for security role references.
 - Security roles are unique.
- Each module listed in the deployment descriptor exists in the archive for EAR files.
- Files for icons, servlets, error and welcome pages listed in the deployment descriptor have corresponding files in the archive for WAR files.
- For EJB modules:
 - All class files referenced in the deployment descriptor exist in the JAR file.
 - Method signatures for enterprise bean home, remote and implementation classes are compliant with the EJB 2.0 specification.

What to do next

If your application module contains EJB modules, generate code for deployment.

Otherwise, you are ready to deploy this application module (or standalone Web module) onto the application server.

Application assembly performance checklist

Application assembly tools are used to build J2EE components and modules into J2EE applications. Generally, assembling consists of defining application components and their attributes including enterprise beans, servlets and resource references. Many of these application configuration settings and attributes play an important role in the run-time performance of the deployed application. Use the following information as a check list of important parameters and advice for finding optimal settings:

- EJB modules
 - Entity bean Bean Cache - Activate at and Bean Cache - Load at settings
 - Method extensions Isolation level and Access intent settings
 - (Container transactions assembly settings)
- Web modules
 - Web modules assembly settings
 - Distributable
 - Reload interval
 - Reload enabled
- Web components
 - Load on startup

Generating code for deployment

Before deploying applications on the server, if the application modules contain EJB modules, you must generate deployment code for the enterprise beans in the application. The Application Assembly Tool (AAT) provides this ability, or you can use the `ejbdeploy` command line tool.

Before you begin

This task assumes you have already assembled an EJB module, added it to an application module, saved the application module, and verified the application module.

Before installing your application in WebSphere Application Server, you must generate deployment code for the application. This step is required for EJB modules and for any Enterprise application (EAR) files that contain EJB modules. During code generation, the Application Assembly Tool invokes the `EJBDeploy` tool to prepare entity bean (JAR) files for deployment in run time environment. To deploy a J2EE application, you can install the application in the administrative console.

The following steps assume that you are using the Application Assembly Tool to generate code for deployment.

Steps for this task

1. Start the Application Assembly Tool (AAT).
2. Open the EAR or JAR file for which you want to generate code for deployment.
3. Click **File > Generate code for deployment** from the menu bar.
4. Specify the options for the server to use for generating code for the application deployment.

Note: For Container managed persistence (CMP) entity beans, if the JAR file that you opened (inputJar file) contains a map and schema document, that schema is used. If the JAR file does not contain a map and schema document, the Application Assembly Tool uses a top-down mapping to generate files that contain mapping and database schema information.

5. Click **Generate Now**.

Review the messaging box for details of any error that might occur.

Note: Do not change the default output file name to be the same as the input filename, as the AAT cannot read and write to the same file name, and therefore, an error will occur.

Results

After deployment code is generated for an application, the deployable archive is renamed with the prefix `Deployed_`.

What to do next

Install the application on your server machine.

Note: Before deploying the application in your run time environment, you might need to set classpaths.

ejbdeploy tool

You can generate code for deployment by either using the Application Assembly Tool (AAT) or by using the Deployment Tool for Enterprise Java Beans (ejbdeploy) from a command prompt. For example, the options that you are able to set in AAT correspond with commands that the EJBDeploy tool uses to generate code for deploying an application.

Syntax

For a detailed list of available options in the EJBDeploy tool, enter `ejbdeploy` from a command prompt.

ejbdeploy syntax — relationship to Application Assembly Tool options

Abstract:

Many options for generating code for deployment in the Application Assembly Tool (AAT) directly correlate with the options in the command line EJBDeploy tool.

Application Assembly Tool options	EJBDeploy tool options
Deployed module location	<code>outputJar</code>
Working Directory	<code>workingDirectory</code>
Dependent classpath	<code>cp</code>
Code generation only	<code>codegen</code>
Verify archive (unchecked)	<code>novalidate</code>
RMIC options	<code>rmic options</code>

Database type	dbvendor
Database name	dbname
Schema name	dbschema

Application Assembly Tool: Resources for learning




Use the following links to find relevant supplemental information about the Application Assembly Tool. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

View links to additional information about:

- Programming specifications
- Administration

Programming specifications

-  **J2EE 1.3 specification** (http://java.sun.com/j2ee/j2ee-1_3-fr-spec.pdf)
-  **EJB specifications** (<http://java.sun.com/products/ejb/docs.html>)
-  **Servlet specifications**
(<http://java.sun.com/products/servlet/download.html>)

Administration

-  **Application Client files**
(<http://developer.java.sun.com/developer/technicalArticles/J2EE/appclient/>)
-  **Connector RAR files** (<http://java.sun.com/j2ee/connector/>)

Chapter 16. Deploying and managing applications

After you develop an enterprise application and configure an application server, you can use the administrative console to install application files on the server and manage the activity of deployed applications.

1. Install your application on your application server.
2. Start and stop applications.
3. Edit the administrative configuration for an application. Go to the settings page for an application, change the values for settings as needed, and click **OK**.
4. Export applications.
5. Export DDL files.
6. Update application binary files.
7. Uninstall applications.

After making changes to administrative configurations of your applications, ensure that you click **Save** on the administrative console taskbar to save the changes.

Enterprise applications

Enterprise applications (or J2EE applications) are applications that conform to the Java 2 Platform, Enterprise Edition, specification.

Enterprise applications can consist of the following:

- Zero or more EJB modules
- Zero or more Web modules
- Zero or more connector modules (packaged in RAR files)
- Zero or more application client modules
- Additional JAR files containing dependent classes or other components required by the application
- Any combination of the above

A J2EE application is represented by, and packaged in, an enterprise archive (EAR) file.

Installing a new application

To install an enterprise application to a WebSphere Application Server configuration, you can use the administrative console or the wsadmin tool. The steps below describe how to use the administrative console to install an application, EJB component, or Web module.

Note: Once you start performing the steps below, click **Cancel** to exit if you decide not to install the application. Do not simply move to another administrative console page without first clicking **Cancel** on an application installation page.

1. Click **Applications > Install New Application** in the console navigation tree. The first of two Preparing for application install pages is shown.
2. On the first Preparing for application install page:
 - a. Specify the full path name of the source application file (.ear file otherwise known as an EAR file). The EAR file that you are installing can be either on the client machine (the machine that runs the Web browser) or on the

server machine (the machine to which the client is connected). If you specify an EAR file on the client machine, then the administrative console uploads the EAR file to the machine on which the console is running and proceeds with application installation. You can also specify a stand-alone WAR or JAR file for installation.

- b. If you are installing a stand-alone WAR file, specify the context root.
 - c. Click **Next**.
3. On the second Preparing for application install page:
 - a. Select whether to generate default bindings. Using the default bindings causes any incomplete bindings in the application to be filled in with default values. Existing bindings are not altered. You can customize default values used in generating default bindings. For example, you can specify JNDI prefix for all the EJB files in EJB modules, default data source and connection factory settings for EJB modules, virtual host for web modules, and so on. ""Preparing for application installation settings" on page 566" describes available customizations and provides sample bindings.
 - b. Click **Next**. The Install New Application pages are now shown. If you chose to generate default bindings, you can proceed to the Summary step (step 23 below). ""Example: Installing an EAR file using the default bindings" on page 569" provides sample steps.
4. On the **Step: Provide options to perform the installation** panel, provide values for the following settings specific to WebSphere Application Server. Default values are used if you do not specify a value.
 - a. For **Pre-compile JSP**, specify whether to precompile JSP files as a part of installation. The default is not to precompile JSP files.
 - b. For **Directory to Install Application**, specify the directory to which the application EAR file will be installed. The default value is the value of `APP_INSTALL_ROOT/cell_name`, where the `APP_INSTALL_ROOT` variable is `install_root/installedApps`; for example, `C:\WebSphere\AppServer\installedApps\cell_name`.

Note: If an installation directory is not specified when an application is installed on a single-server (base) configuration, the application is installed in `APP_INSTALL_ROOT/base_cell_name`. When the base server is made a part of a Network Deployment configuration (using the `addNode` utility), the cell name of the new configuration becomes the cell name of the deployment manager node. If the `-includeapps` option is used for the `addNode` utility, then the applications that are installed prior to the `addNode` operation still use the installation directory `APP_INSTALL_ROOT/base_cell_name`. However, an application that is installed after the base server is added to the network configuration uses the default installation directory `APP_INSTALL_ROOT/network_cell_name`. To move the application to the `APP_INSTALL_ROOT/network_cell_name` location upon running the `addNode` operation, you should explicitly specify the installation directory as `${APP_INSTALL_ROOT}/${CELL}` during installation. In such a case, the application files can always be found under `APP_INSTALL_ROOT/current_cell_name`.
 - c. For **Distribute Application**, specify whether WebSphere Application Server expands or deletes application binaries in the installation destination. The default is to enable application distribution. As a result, when you save changes in the console, application binaries for newly installed applications are expanded to the directory specified. The binaries are also deleted when you uninstall and save changes to the configuration. If you disable this

option, then you must ensure that the application binaries are expanded appropriately in the destination directories of all nodes where the application is expected to run.

- d. For **Use Binary Configuration**, specify whether the application server uses the binding, extensions, and deployment descriptors located with the application deployment document, the deployment.xml file (default), or those located in the EAR file. The default is not to use the binary configuration.
- e. For **Deploy EJBs**, specify whether the EJBDeploy tool runs during application installation. The tool generates code needed to run EJB files. The default is not to run the EJBDeploy tool. You must enable this setting if the EAR file was assembled using the Application Assembly (AAT) tool and the EJBDeploy tool was not run during assembly, if the EAR file was not assembled using the AAT tool, or if the EAR file was assembled using versions of the AAT tool previous to Version 5. Note that enabling this setting might cause the installation program to run for several minutes.
- f. For **Application Name**, name the application. Application names must be unique within a cell and cannot contain characters that are not allowed in object names.
- g. For **Create MBeans for Resources**, specify whether to create MBeans for various resources (such as servlets or JSP files) within an application when the application is started. The default is to create MBean instances.
- h. For **Enable class reloading**, specify whether to enable class reloading when application files are updated. The default is not to enable class reloading.
- i. For **Reload Interval**, specify the number of seconds to scan the application's file system for updated files. The default is the value of the reload interval attribute in the IBM extension (META-INF/ibm-application-ext.xmi) file of the EAR file. This setting takes effect only if class reloading is enabled.

The reload interval specified here overrides the value specified in the IBM extensions for each Web module in the EAR file (which in turn overrides the reload interval specified in the IBM extensions for the application in the EAR file).

- 5. If your application uses EJB modules, on the **Step: Provide JNDI Names for Beans** panel, specify a JNDI name for each enterprise bean in every EJB module. You must specify a JNDI name for every enterprise bean defined in the application. For example, for the EJB module MyBean.jar, specify MyBean.
- 6. If your application uses EJB modules that contain Container Managed Persistence (CMP) beans that are based on the EJB 1.x specification, for **Step: Provide default datasource mapping for modules containing 1.x entity beans**, specify a JNDI name for the default data source for the EJB modules. The default data source for the EJB modules is optional if data sources are specified for individual CMP beans.
- 7. If your application has CMP beans that are based on the EJB 1.x specification, for **Step: Map datasources for all 1.x CMP**, specify a JNDI name for data sources to be used for each of the 1.x CMP beans. The data source attribute is optional for individual CMP beans if a default data source is specified for the EJB module that contains CMP beans. If neither a default data source for the EJB module nor a data source for individual CMP beans are specified, then a validation error displays after you click **Finish** (step 23) and the installation is cancelled.

8. If your application defines EJB references, for **Step: Map EJB references to beans**, specify JNDI names for enterprise beans that represent the logical names specified in EJB references. Each EJB reference defined in the application must be bound to an EJB file before clicking **Finish** on the Summary panel.
9. If your application defines resource references, for **Step: Map resource references to resources**, specify JNDI names for the resources that represent the logical names defined in resource references. Each resource reference defined in the application must be bound to a resource defined in your WebSphere Application Server configuration before clicking on **Finish** on the Summary panel.
10. If your application uses Web modules, for **Step: Map virtual hosts for web modules**, select a virtual host from the list that should map to a Web module defined in the application. The port number specified in the virtual host definition is used in the URL that is used to access artifacts such as servlets and JSP files in the Web module. Each Web module must have a virtual host to which it maps. Not specifying all needed virtual hosts will result in a validation error displaying after you click **Finish** on the Summary panel.
11. On the **Step: Map modules to application servers** panel, for every module select a target server or a cluster from the **Clusters and Servers** list. Place a checkmark in the check box beside **Module** to select all of the application modules or select individual modules.
12. If the application has security roles defined in its deployment descriptor then, for **Step: Map security roles to users/groups**, specify users and groups that are mapped to each of the security roles. Place a checkmark in the check box beside **Role** to select all of the roles or select individual roles. For each role, you can specify if predefined users such as **Everyone** or **All Authenticated users** are mapped to it. To select specific users or groups from the user registry:
 - a. Select a role and click **Lookup users** or **Lookup groups**.
 - b. On the Lookup users/groups panel shown, enter search criteria to extract a list of users or groups from the user registry.
 - c. Select individual users or groups from the results displayed.
 - d. Click **OK** to map the selected users or groups to the role selected on the **Step: Map security roles to users/groups** panel.
13. If the application has RunAs roles defined in its deployment descriptor, for **Step: Map RunAs roles to user**, specify the RunAs user name and password for every RunAs role. RunAs roles are used by enterprise beans that must run as a particular role while interacting with another enterprise bean. Place a checkmark in the check box beside **Role** to select all of the roles or select individual roles. After selecting a role, enter values for the user name, password, and verify password and click **Apply**.
14. If your application contains EJB 1.x CMP beans that do not have method permissions defined for some of the EJB methods, for **Step: Ensure all unprotected 1.x methods have the correct level of protection**, specify if you want to leave such methods unprotected or assign protection with deny all access.
15. If your application contains message driven enterprise beans, for **Step: Provide Listener Ports for messaging beans**, provide a listener port name for every message driven bean. If a name is not specified for each bean, then a validation error displays after you click on **Finish** on the Summary panel.
16. If your application uses EJB modules that contain CMP beans that are based on the EJB 2.0 specification, for **Step: Provide default datasource mapping for**

modules containing 2.0 entity beans, specify a JNDI name for the default data source and the type of resource authorization to be used for the default data source for the EJB modules. The default data source for EJB modules is optional if data sources are specified for individual CMP beans.

17. If your application has CMP beans that are based on the EJB 2.0 specification, on the **Step: Map datasources for all 2.0 CMP** panel, for each of the 2.0 CMP beans specify a JNDI name and the type of resource authorization for data sources to be used. The data source attribute is optional for individual CMP beans if a default data source is specified for the EJB module that contains CMP beans. If neither a default data source for the EJB module nor a data source for individual CMP beans are specified, then a validation error is shown after you click **Finish** and installation is cancelled.
18. If your application contains EJB 2.0 CMP beans that do not have method permissions defined in the deployment descriptors for some of the EJB methods, on the **Step: Ensure all unprotected 2.0 methods have the correct level of protection** panel, specify whether you want to assign a specific role to the unprotected methods, add the methods to the exclude list, or mark them as unchecked. Methods added to the exclude list are marked as uncallable. For methods marked unchecked no authorization check is performed prior to their invocation.
19. If the **Deploy EJBs** setting is enabled on the **Provide options to perform the installation** panel, then you can specify options for the EJBDeploy tool on the **Step: Provide options to perform the EJB Deploy** panel. On this panel, you can specify extra classpath, rmic options, database types, and database schema names to be used while running the EJBDeploy tool. The tool is run on the EAR file during installation after you click **Finish**.
20. If your application contains resource environment references, for **Step: Mapping Resource Environment References to Resources**, specify JNDI names of resources that map to the logical names defined in resource environment references. If each resource environment reference does not have a resource associated with it, a validation error is shown after you click **Finish**.
21. If your application defines **Run-As Identity** as *System Identity*, for **Step: Replacing RunAs System to RunAs Roles**, you can optionally change it to *Run-As role* and specify a user name and password for the RunAs role specified. Selecting *System Identity* implies that the invocation is done using the WebSphere Application Server security server ID and should be used with caution as this ID has more privileges.
22. If your application has resource references that map to resources that have an Oracle database doing back-end processing, for **Step: Specify the isolation level for Oracle type provider**, specify or correct the isolation level to be used for such resources when used by the application. Oracle databases support ReadCommitted and Serializable isolation levels only.
23. On the Summary panel, verify the cell, node, and server onto which the application modules will install. Beside the **Cell/Node/Server** option, click **Click here** and verify the settings. Then click **Finish**.

Note: After clicking **Finish**, if you receive an OutOfMemory exception and the source application file does not install, your system might not have enough memory or your application might have too many modules in it to install successfully onto the server. If lack of system memory is not the cause of the exception, package your application again so the .ear file has fewer modules. If lack of system memory and the number of modules are not the cause of the

exception, check the options you specified on the Java Virtual Machine page of the application server running the administrative console. Then, try installing the application file again.

24. Associate any shared libraries that the application needs to the application.
25. Click **Save** on the administrative console taskbar to save the changes to your configuration. The application is registered with the administrative configuration and application files are copied to the target directory, which is *install_root/installedApps/cell_name* by default or the directory that you designate. For the single-server (base) installation, application files are copied to the destination directory when you click **Save**; for the Network Deployment installation, files are copied to remote nodes when the configuration on the deployment manager synchronizes with the configuration on individual nodes.
26. Test the application. For example, point a Web browser at the URL for the deployed application and examine the performance of the application. If necessary, update the application.

Preparing for application installation settings

Use this page to install an application (EAR file) or module (JAR or WAR file). To view this administrative console page, click **Applications > Install New Application**.

Follow the steps on this page to install an application or module. You must complete, at minimum, the first step; you must complete some or all of the later steps, depending on whether you are installing an application, EJB module, or Web module.

Path

Specifies the fully qualified path to the .ear, .jar, or .war file for the enterprise application.

Use **Local path** if the browser and application files are on the same machine (whether or not the server is on that machine, too).

Use **Server path** if the application file resides on any node in the current cell context. You can browse the entire file system of a node if the node agent or deployment manager is running on that selected node. Only .ear, .jar, or .war files are shown during the browsing.

During application installation, application files are typically uploaded from a client machine running the browser to the server machine running the administrative console, where they are deployed. In such cases, the Web browser running the administrative console is used to select EAR, WAR, or JAR modules to upload to the server machine.

In some cases, however, the application files reside on the file system of any of the nodes in a cell. To have the application server install these files, use the **Server path** option.

You can also use this option to specify an application file already residing on the machine running the application server. For example, the field value on Windows NT might be C:\WebSphere\AppServer\installableApps\test.ear. If you are installing a stand-alone WAR module, then you also must specify the context root.

Context Root

Specifies the context root of the Web application (WAR).

This field is used only to install a stand-alone WAR file. The context root is combined with the defined servlet mapping (from the WAR file) to compose the full URL that users type to access the servlet. For example, if the context root is /gettingstarted and the servlet mapping is MySession, then the URL is http://host:port/gettingstarted/MySession.

Generate Default Bindings

Specifies whether to generate default bindings. If you place a checkmark in the check box, then any incomplete bindings in the application are filled in with default values. Existing bindings are not altered.

By choosing this option, you can directly jump to the Summary step and install the application if none of the steps have a red asterisk (*) next to them. A red asterisk denotes that the step has incomplete data and requires a valid value. On the Summary panel, verify the cell, node and server on which the application is installed.

Bindings are generated as follows:

- EJB JNDI names are generated of the form *prefix/ejb-name*. The default prefix is *ejb*, but can be overridden. The *ejb-name* is as specified in the deployment descriptors `<ejb-name>` tag.
- EJB references are bound as follows: If an `<ejb-link>` is found, it is honored. Otherwise, if a unique enterprise bean is found with a matching home (or local home) interface as the referenced bean, the reference is resolved automatically.
- Resource reference bindings are derived from the `<res-ref-name>` tag. Note that this action assumes that the `java:comp/env` name is the same as the resource global JNDI name.
- Connection factory bindings (for EJB 2.0 JAR files) are generated based on the JNDI name and authorization information provided. This action results in default connection factory settings for each EJB 2.0 JAR file in the application being installed. No bean-level connection factory bindings are generated.
- Data source bindings (for EJB 1.1 JAR files) are generated based on the JNDI name, data source user name password options. This results in default data source settings for each EJB JAR file. No bean-level data source bindings are generated.
- Message Driven Bean (MDB) listener ports are derived from the MDB `<ejb-name>` tag with the string `Port` appended.
- For `.war` files, the virtual host is set as `default_host` unless otherwise specified.

The default strategy suffices for most applications or at least for most bindings in most applications. However, it does not work if:

- You want to explicitly control the global JNDI names of one or more EJB files.
- You need tighter control of data source bindings for CMPs. That is, you have multiple data sources and need more than one global data source.
- You must map resource references to global resource JNDI names that are different from the `java:comp/env` name.

In such cases, you can alter the behavior with an XML document (a custom strategy). Use the **Specific bindings file** field to specify a custom strategy and see the field's help for examples.

Prefixes

Specifies prefixes to use for generated JNDI names.

Override

Specifies whether to override existing bindings.

If this check box is checked, the existing bindings are overridden by the generated ones.

EJB 1.1 CMP bindings

Specifies the default data source JNDI name.

If the **Default Bindings for EJB 1.1 CMPs** radio button is selected, specify the JNDI name for the default data source to be used with the CMP 1.1 beans. Also specify the user ID and password for this default data source.

Connection Factory Bindings

Specifies the default data source JNDI name.

If the **Default connection factory bindings** radio button is selected, specify the JNDI name for the default data source to be used with the bindings. Also specify the resource authorization.

Virtual Host

Specifies the virtual host for WAR modules.

Specific bindings file

Specifies a bindings file that overrides the default binding.

Alter the behavior of the default binding with an XML document (*aka* custom strategy). Custom strategies extend the default strategy so you only need to customize those areas where the default strategy is insufficient. That is, you only need to describe how you want to change the bindings generated by the default strategy; you do not have to define bindings for the entire application.

Brief examples of how to override various aspects of the default bindings generator follow:

Controlling an EJB JNDI name

```
<?xml version="1.0"?>
<!DOCTYPE dfltbndngs SYSTEM "dfltbndngs.dtd">
<dfltbndngs>
  <module-bindings>
    <ejb-jar-binding>
      <jar-name>helloEjb.jar</jar-name>
<!-- this name must match the module name in the .ear file -->
    <ejb-bindings>
      <ejb-binding>
        <ejb-name>HelloEjb</ejb-name>
<!-- this must match the <ejb-name> entry in the EJB jar DD -->
        <jndi-name>com/acme/ejb/HelloHome</jndi-name>
      </ejb-binding>
    </ejb-bindings>
  </ejb-jar-binding>
</module-bindings>
</dfltbndngs>
```

Setting the connection factory binding for an EJB JAR file

```
<!DOCTYPE dfltbndngs SYSTEM "dfltbndngs.dtd">
<dfltbndngs>
  <module-bindings>
    <ejb-jar-binding>
      <jar-name>yourEjb20.jar</jar-name>
      <connection-factory>
        <jndi-name>eis/jdbc/YourData_CMP</jndi-name>
        <res-auth>Container</res-auth>
      </connection-factory>
    </ejb-jar-binding>
  </module-bindings>
</dfltbndngs>
```

Setting the connection factory binding for an EJB file

```

<?xml version="1.0">
<!DOCTYPE dfltbndngs SYSTEM "dfltbndngs.dtd">
<dfltbndngs>
  <module-bindings>
    <ejb-jar-binding>
      <jar-name>yourEjb20.jar</jar-name>
      <ejb-bindings>
        <ejb-binding>
          <ejb-name>YourCmp20</ejb-name>
<!-- this matches the ejb-name tag in the DD -->
          <connection-factory>
            <jndi-name>eis/jdbc/YourData_CMP</jndi-name>
            <res-auth>PerConnFact</res-auth>
          </connection-factory>
        </ejb-binding>
      </ejb-bindings>
    </ejb-jar-binding>
  </module-bindings>
</dfltbndngs>

```

Overriding a Resource Ref Binding from a WAR, EJB JAR file, or J2EE client JAR file

Example code for overriding a Resource Ref Binding from a WAR file follows. Use similar code to override a Resource Ref Binding from an EJB JAR file or a J2EE client JAR file.

```

<?xml version="1.0"?>
<!DOCTYPE dfltbndngs SYSTEM "dfltbndngs.dtd">
<dfltbndngs>
  <module-bindings>
    <war-binding>
      <jar-name>hello.war</jar-name>
      <resource-ref-bindings>
        <resource-ref-binding>
          <!-- the following must match the resource-ref in the DD -->
          <resource-ref-name>jdbc/MyDataSrc</resource-ref-name>
          <jndi-name>war/override/dataSource</jndi-name>
        </resource-ref-binding>
      </resource-ref-bindings>
    </war-binding>
  </module-bindings>
</dfltbndngs>

```

Overriding MDB JMS listener ports

```

<?xml version="1.0"?>
<!DOCTYPE dfltbndngs SYSTEM "dfltbndngs.dtd">
<dfltbndngs>
  <module-bindings>
    <ejb-jar-binding>
      <jar-name>YourEjbJar.jar</jar-name>
      <ejb-bindings>
        <ejb-binding>
          <ejb-name>YourMDB</ejb-name>
          <listener-port>yourMdbListPort</listener-port>
        </ejb-binding>
      </ejb-bindings>
    </ejb-jar-binding>
  </module-bindings>
</dfltbndngs>

```

Example: Installing an EAR file using the default bindings

An example of a simple .ear file installation using the default bindings follows:

1. Go to the Preparing for application install pages. Click **Applications > Install an Application** in the console navigation tree.
2. For **Path**, specify the full path name of the .ear file. For this example, the base file name is my_appl.ear and the file resides on a server at C:\sample_apps.
 - a. Select the **Server path** radio button and click **Browse**.
 - b. On the Browse Remote Filesystems page, click on the name of the node that holds the my_appl.ear file, **C:\, sample_apps, my_appl.ear**, and then **OK**.
3. Now that a value is given for **Path**, on the first Preparing for application install page, click **Next**.
4. On the second Preparing for application install page, place a checkmark beside the **Generate Default Bindings** check box and click **Next**. Using the default bindings causes any incomplete bindings in the application to be filled in with default values. Existing bindings are not changed. By choosing this option, you can directly jump to the Summary step.
5. On the Install New Application page, click on **Summary**, the last step.
6. On the Summary panel, verify the cell, node, and server onto which the application files will install.
 - a. Beside the **Cell/Node/Server** option, click **Click here**.
 - b. On the **Map modules to application servers** panel, select the server onto which the application files will install from the **Clusters and Servers** list, place a checkmark in the check box beside **Module** to select all of the application modules, and click **Next**.

Because my_appl.ear does not require any additional settings to complete an installation, the Summary panel displays again.

7. On the Summary panel, click **Finish**.

Enterprise application collection

Use this page to view and manage enterprise applications.

To view this administrative console page, click **Applications > Enterprise Applications**.

Name

Specifies the name of the installed (or deployed) application. Application names must be unique within a cell and cannot contain characters that are not allowed in object names.

Status

Indicates whether the application deployed on the application server is started, stopped, or unavailable.

Enterprise application settings

Use this page to configure an enterprise application.

To view this administrative console page, click **Applications > Enterprise Applications > *application_name***.

Name

Specifies a logical name for the application. Application names must be unique within a cell and cannot contain characters that are not allowed in object names.

Data type

String

Starting Weight

Specifies the order in which applications are started when the server starts. The application with the lowest starting weight is started first.

Data type	Integer
Default	1
Range	0 to 100

Application Binaries

Specifies the directory to which the application EAR file will be installed. The default value is the value of `APP_INSTALL_ROOT/cell_name`, where the `APP_INSTALL_ROOT` variable is `install_root/installedApps`; for example, `C:\WebSphere\AppServer\installedApps\cell_name`.

You can specify an absolute path or use a pathmap variable such as `${MY_APPS}`. You can use a pathmap variable in any installation though it is particularly needed when installing an application on a cluster with members on heterogeneous nodes because, in such cases, there might not be a single way to specify an absolute path. A WebSphere Application Server variable `${CELL}` that denotes the current cell name can also be in the pathmap variable; for example, `${MY_APP}/${CELL}`.

Data type	String
Units	Full path name

Use Metadata From Binaries

Specifies whether the application server uses the binding, extensions, and deployment descriptors located with the application deployment document, the `deployment.xml` file (default), or those located in the enterprise application resource (EAR) file.

Data type	Boolean
Default	true

Enable Distribution

Specifies whether WebSphere Application Server expands or deletes application binaries in the installation destination. The default is to enable application distribution. Application binaries for installed applications are expanded to the directory specified. The binaries are also deleted when you uninstall and save changes to the configuration. If you disable this option, then you must ensure that the application binaries are expanded appropriately in the destination directories of all nodes where the application runs.

Data type	Boolean
Default	true

ClassLoader Mode

Specifies whether the class loader searches in the parent class loader or in the application class loader first to load a class. The standard for JDK class loaders and WebSphere Application Server class loaders is `PARENT_FIRST`. By specifying `PARENT_LAST`, your application can override classes contained in the parent class loader, but this action can potentially result in `ClassCastException` or `LinkageErrors` if you have mixed use of overridden classes and non-overridden classes.

The options are `PARENT_FIRST` and `PARENT_LAST`. The default is to search in the parent class loader before searching in the application class loader to load a class.

Data type	String
Default	PARENT_FIRST

WAR Classloader Policy

Specifies whether to use a single class loader to load all WAR files of this application or to use a different class loader for each WAR file. The options are APPLICATION and MODULE. The default is to use a separate class loader to load each WAR file.

Data type	String
Default	MODULE

Create MBeans for Resources

Specifies whether to create MBean files for various resources (such as servlets or JSP files) within an application.

Data type	Boolean
Default	true

Reload Enabled

Specifies whether to enable class reloading when application files are updated.

Data type	Boolean
Default	true

Reload Interval

Specifies the number of seconds to scan the application's file system for updated files. The default is the value of the reload interval attribute in the IBM extension (META-INF/ibm-application-ext.xmi) file of the EAR file. This setting takes effect only if class reloading is enabled.

The reload interval specified here overrides the value specified in the IBM extensions for each Web module in the EAR file (which in turn overrides the reload interval specified in the IBM extensions for the application in the EAR file).

Data type	Integer
Units	Seconds

Target mapping collection

Use this page to manage mappings of deployed applications or modules to servers or clusters.

To view this administrative console page, click **Applications > Enterprise Applications > *application_name* > Target Mappings**.

Target:

States the name of the target server or cluster to which the application or module maps. You specify the target on the Map modules to application servers page accessed from the settings for an application.

Node:

Specifies the node name if the target is a server.

Status:

Indicates whether the status of the application running on the target server or cluster is started, stopped or unavailable.

Target mapping settings:

Use this page to map a deployed application or module to a server or cluster.

To view this administrative console page, click **Applications > Enterprise Applications > application_name > Target Mappings > target_name**.

Target:

States the name of the target server or cluster to which the application or module maps. You specify the target on the Map modules to application servers page accessed from the settings for an application.

Data type String

Enabled:

Indicates whether the application modules installed on the target server are started (or enabled) when the server starts. This sets the initial state of application modules. A true value indicates that the corresponding modules are enabled and thus are accessible when the server starts. A false value indicates that the corresponding modules are not enabled and thus are not accessible when the server starts.

Data type Boolean
Default true

Starting and stopping applications

You can start an application that is not running (has a status of *Stopped*) or stop an application that is running (has a status of *Started*).

1. Go to the Enterprise Applications page. Click **Applications > Enterprise Applications** in the console navigation tree.
2. Check the check box for the application you want started or stopped.
3. Click a button:

Option	Description
Start	Runs the application and changes the state of the application from <i>Stopped</i> to <i>Started</i> .
Stop	Stops the processing of the application and changes the state of the application from <i>Started</i> to <i>Stopped</i> .

To restart a running application, place a check mark in the check box for the application you want to restart, click **Stop** and then click **Start**.

The status of the application changes and a message stating that the application started or stopped displays at the top the page.

Exporting applications

You can export an enterprise application to a location of your choice. Exporting applications enables you to back up your applications and preserve binding information for the applications. You might export your applications before updating installed applications or migrating to a later version of the WebSphere Application Server product.

1. Click **Applications > Enterprise Applications** in the administrative console navigation tree to access the Enterprise Applications page.
2. Place a checkmark in the check box beside the application and click **Export**.
3. On the Export Application EAR Files page, click on the link to download the exported EAR file.
4. Use the browser dialogue to specify a location at which to save the exported EAR file and click **OK**.

The file containing binding information is exported to the specified node and directory, and has the name *enterprise_application_name.ear*.

Exporting DDL files

You can export the DDL files (Table.ddl) in the EJB modules of the application to a location of your choice.

1. Click **Applications > Enterprise Applications** in the administrative console navigation tree to access the Enterprise Applications page.
2. Place a checkmark in the check box beside the application and click **Export DDL**. If the application has no DDL files in any of its EJB modules, then the message *No DDL files were found* displays at the top of the page. If the application has DDL files in its EJB modules, then a page displays listing DDL files in the format *appname.ear/_module.jar_Table.ddl*.
3. Click on a file in the list to download the file to your machine.

Updating applications

You can update an application deployed on a server. The steps below describe how to update a deployed application using the administrative console.

Note: You can also update applications using the wsadmin tool, which provides updating capabilities identical to those available using the administrative console. Further, in some situations, you can update applications without needing to restart the applications.

1. Update the contents of the application and reassemble it, using the Application Assembly Tool. Typical tasks include adding or editing assembly properties, adding or importing modules into an application, and adding enterprise beans, Web components, and files.
2. Go to the Applications page of the administrative console. Click **Applications > Enterprise Applications** in the console navigation tree.
3. Back up the application. Place a checkmark in the check box beside the application you want uninstalled and click **Export** to export the application to an EAR file and preserve the binding information.
4. With a checkmark beside the application, click **Update**. The binding information of the updated (new) version of the application merges with the

binding information from the installed (old) version. Then, the older version uninstalls from the configuration and the new version installs.

5. Complete the steps in the Preparing for application update page and the pages that follow it. See information on installing applications and on the settings page for application installation for guidance. Note that the installation steps have the merged binding information from the new version and the old version. If the new version has bindings for application artifacts such as EJB JNDI names, EJB references or resource references, then those bindings will be part of the merged binding information. If new bindings are not present, then bindings are taken from the installed (old) version. If bindings are not present in the old version and if the default binding generation option is enabled, then the default bindings will be part of the merged binding information. You can select whether to ignore bindings in the old version or ones in the new version.
6. Map the installed application or module to servers or clusters. Use the Map modules to application servers page of the Install New Application pages displayed during updating the application. Or, after updating the application, use the Map modules to application servers page accessed from the Enterprise Applications page.
 - a. Go to the Map modules to application servers page. Click **Applications > Enterprise Applications** in the console navigation tree, click the application name, and then click **Map modules to application servers**.
 - b. Specify the application server where you want to install modules contained in your application and click **OK**.
7. Click **Save** on the administrative console taskbar to save the changes to your configuration. In the single server (base) product, after you click **Save** the old version of the application is uninstalled and the new version is installed into the configuration. The application binaries for the old version are deleted from the destination directory and the new binaries are copied to the directory. In the Network Deployment product, the old application files are deleted and new files are copied when the configuration on the deployment manager synchronizes with the configuration on the node where the application is installed. If the application is running when you update it, the application stops running before its files are copied to the destination directory of the node and restarts after the copy operation completes. Thus, the application is unavailable on the node during the time the node is synchronizing its configuration with the deployment manager.
8. Restart the application so the changes take effect. If the application is updated while it is running, WebSphere Application Server stops the application, updates the application logic and restarts the application.
 - a. Click **Applications > Enterprise Applications** in the console navigation tree to go to the Enterprise Applications page.
 - b. Check the check box for the updated application.
 - c. Click **Start**.
9. Optional: If the application you are updating is deployed on a server that has its application class loader policy set to **SINGLE**, restart the server.

Hot deployment and dynamic reloading

You can make various changes to applications and their contents without having to stop the server and start it again. Making these types of changes is known as *hot deployment and dynamic reloading*.

Hot deployment is the process of adding new components (such as WAR files, EJB Jar files, enterprise Java beans, servlets, and JSP files) to a running server without having to stop the application server process and start it again.

Dynamic reloading is the ability to change an existing component without needing to restart the server in order for the change to take effect. Dynamic reloading involves:

- Changes to the implementation of a component of an application, such as changing the implementation of a servlet
- Changes to the settings of the application, such as changing the deployment descriptor for a Web module

If the application you are updating is deployed on a server that has its application class loader policy set to `SINGLE`, you might not be able to dynamically reload your application. At minimum, you must restart the server after updating your application.

1. Locate your expanded application files. The application files are in the directory you specified when installing the application or, if you did not specify a custom target directory, are in the default target directory, *install_root/installedApps/cell_name*. Your EAR file, `${APP_INSTALL_ROOT}/cell_name/application_name.ear`, points to the target directory. The `variables.xml` file for the node defines `${APP_INSTALL_ROOT}`.

It is important to locate the expanded application files because, as part of installing applications, a WebSphere application server unjars portions of the EAR file onto the file system of the computer that will run the application. These expanded files are what the server looks at when running your application.

If you cannot locate the expanded application files, look at the `binariesURL` attribute in the `deployment.xml` file for your application. The attribute designates the location the run time uses to find the application files.

For the remainder of this information on hot deployment and dynamic reloading, *application_root* represents the root directory of the expanded application files.

2. Locate application metadata files. The metadata files include the deployment descriptors (`web.xml`, `application.xml`, `ejb-jar.xml`, and the like), the bindings files (`ibm-web-bnd.xmi`, `ibm-app-bnd.xmi`, and the like), and the extensions files (`ibm-web-ext.xmi`, `ibm-app-ext.xmi`, and the like).

Metadata XML files for an application can be loaded from one of two locations. The metadata files can be loaded from the same location as the application binary files (such as *application_root/META-INF*) or they can be loaded from the WebSphere configuration tree,

`${CONFIG_ROOT}/cells/cell_name/applications/application_EAR_name/deployments/application_`

The value of the `useMetadataFromBinary` flag specified during application installation controls which location is used. If specified, the metadata files are loaded from the same location as the application binary files. If not specified, the metadata files are loaded from the application deployment folder in the configuration tree.

For the remainder of this information, *metadata_root* represents the location of the metadata files for the specified application or module.

3. CAUTION: If you are running WebSphere Application Server on a group of machines using Network Deployment and you are changing an application on a particular node, disable automatic synchronization.

- a. Click **System Administration > Node Agents** in the administrative console navigation tree, click on a node agent name, and then click **File Synchronization Service**.
- b. On the File Synchronization Service page, remove the checkmark from the check box for **Automatic Synchronization** and click **OK**.

When you run WebSphere Application Server on a group of machines using Network Deployment and you change a file on the disk in the expanded application directory for a particular node, you can lose those changes the next time node synchronization occurs. In the Network Deployment environment, the configuration stored by the deployment manager is the master copy and any changes detected between that master copy and the copy on a particular machine trigger the master copy to be downloaded to the node.

4. Change or add the following components or modules as needed:
 - Application files
 - WAR files
 - EJB Jar files
 - HTTP plug-in configuration files
5. For changes to take effect, you might need to start, stop, or restart an application. "Starting and stopping applications" on page 573 provides information on using the administrative console to start, stop, or restart an application. "Example: Starting an application using wsadmin" and "Example: Stopping running applications on a server using wsadmin" provide information on using the wsadmin scripting tool.
6. If you disabled automatic synchronization in step 3, return to the File Synchronization Service page, enable **Automatic Synchronization**, and click **OK**.

Changing or adding application files

You can change or add application files on application servers without having to stop the server and start it again. This file describes--

- Updating an existing application on a running server (providing a new EAR file)
- Adding a new application to a running server
- Removing an existing application from a running server
- Adding a new EJB or Web module to an existing, running application
- Changing the application.xml file for an application
- Changing the ibm-app-ext.xmi file for an application
- Changing the ibm-app-bnd.xmi file for an application
- Changing a non-module Jar file contained in the EAR file
- Update an existing application on a running server (providing a new EAR file). Reinstall an updated application using the administrative console or the `wsadmin $AdminApp install` command with the `-update` option

Both reinstallation methods enable you to update an existing application using any of the other steps listed in this file, including changing classes, adding modules, removing modules, changing modules, or changing metadata files. The application reinstallation methods detect the changes in your application and prompt you for additional binding data that might be needed to install the application. The reinstallation process automatically stops and restarts your application on the appropriate servers.

Hot deployment:	Yes
Dynamic reloading:	Yes

- Add a new application to a running server. Install an application using the administrative console or the `wsadmin install` command.

Hot deployment: Yes
Dynamic reloading: No

- Remove an existing application from a running server. Stop the application and then uninstall it from the server. Use the administrative console to stop the application and then uninstall it. Or run the `wasadmin stopApplication` command and then the `uninstall` command.

Hot deployment: Yes
Dynamic reloading: No

- Add a new EJB or Web module to an existing, running application.
 1. Update the application files in the *application_root* location.
 2. Restart the application. Use the administrative console to restart the application. Or run the `wasadmin stopApplication` and `startApplication` commands.

Hot deployment: Yes
Dynamic reloading: No

- Change the `application.xml` file for an application. Restart the application. Automatic reloading will not detect the change. Use the administrative console to restart the application. Or run the `wasadmin stopApplication` and `startApplication` commands.

Hot deployment: Not applicable
Dynamic reloading: Yes

- Change the `ibm-app-ext.xmi` file for an application. Restart the application. Automatic reloading will not detect the change. Use the administrative console to restart the application. Or run the `wasadmin stopApplication` and `startApplication` commands.

Hot deployment: Not applicable
Dynamic reloading: Yes

- Change the `ibm-app-bnd.xmi` file for an application. Restart the application. Automatic reloading will not detect the change. Use the administrative console to restart the application. Or run the `wasadmin stopApplication` and `startApplication` commands.

Hot deployment: Not applicable
Dynamic reloading: Yes

- Change a non-module Jar file contained in the EAR file.
 1. Update the non-module Jar file in the *application_root* location.
 2. If automatic reloading is not enabled, restart the application. Use the administrative console to restart the application. Or run the `wasadmin stopApplication` and `startApplication` commands. If automatic reloading is enabled, you do not need to take further action. Automatic reloading will detect the change.

Hot deployment: Yes

Dynamic reloading: Yes

Changing or adding WAR files

You can change WAR files on application servers without having to stop the server and start it again. This file describes--

- Changing an existing JSP file
- Adding a new JSP file to an existing application
- Changing an existing servlet class (editing and recompiling)
- Changing a dependent class of an existing servlet class
- Adding a new servlet using the Invoker (Serve Servlets by class name) facility or adding a dependent class to an existing application
- Adding a new servlet, including a new definition of the servlet in the web.xml deployment descriptor for the application
- Changing the web.xml file of a WAR file
- Changing the ibm-web-ext.xmi file of a WAR file
- Changing the ibm-web-bnd.xmi file of a WAR file
- Change an existing JSP file. Place the changed JSP file directly in the *application_root/module_name* directory or the appropriate subdirectory. The change will be automatically detected and the JSP will be recompiled and reloaded.

Hot deployment: Not applicable

Dynamic reloading: Yes

- Add a new JSP file to an existing application. Place the new JSP file directly in the *application_root/module_name* directory or the appropriate subdirectory. The new file will be automatically detected and compiled on the first request to the page.

Hot deployment: Yes

Dynamic reloading: Yes

- Change an existing servlet class (editing and recompiling).
 1. Place the new version of the servlet .class file directly in the *application_root/module_name/WEB-INF/classes* directory. If the .class file is part of a Jar file, you can place the new version of the Jar file directly in *application_root/module_name/WEB-INF/lib*. In either case, the change will be detected, the Web application will be shut down and reinitialized, picking up the new class.
 2. If automatic reloading is not enabled, restart the application. Use the administrative console to restart the application. Or run the wasadmin stopApplication and startApplication commands. If automatic reloading is enabled, you do not need to take further action. Automatic reloading will detect the change.

Hot deployment: Not applicable

Dynamic reloading: Yes

- Change a dependent class of an existing servlet class.
 1. Place the new version of the dependent .class file directly in the *application_root/module_name/WEB-INF/classes* directory. If the .class file is part of a Jar file, you can place the new version of the Jar file directly in

application_root/module_name/WEB-INF/lib. In either case, the change will be detected, the Web application will be shut down and reinitialized, picking up the new class.

2. If automatic reloading is not enabled, restart the application. Use the administrative console to restart the application. Or run the `wasadmin stopApplication` and `startApplication` commands. If automatic reloading is enabled, you do not need to take further action. Automatic reloading will detect the change.

Hot deployment: Not applicable
Dynamic reloading: Yes

- Add a new servlet using the Invoker (Serve Servlets by class name) facility or adding a dependent class to an existing application.
 1. Place the new .class file directly in the *application_root/module_name/WEB-INF/classes* directory. If the .class file is part of a Jar file, you can place the new version of the Jar file directly in *application_root/module_name/WEB-INF/lib*. In either case, the change will be detected, the Web application will be shut down and reinitialized, picking up the new class. This case is treated the same as changing an existing class. The difference is that adding the servlet or class does not immediately cause the Web application to reload because the class has never been loaded before. The class simply becomes available for execution.
 2. If automatic reloading is not enabled, restart the application. Use the administrative console to restart the application. Or run the `wasadmin stopApplication` and `startApplication` commands. If automatic reloading is enabled, you do not need to take further action. Automatic reloading will detect the change.

Hot deployment: Yes
Dynamic reloading: Not applicable

- Add a new servlet, including a new definition of the servlet in the `web.xml` deployment descriptor for the application. Place the new .class file directly in the *application_root/module_name/WEB-INF/classes* directory. If the .class file is part of a Jar file, you can place the new version of the Jar file directly in *application_root/module_name/WEB-INF/lib*.
You can edit the `web.xml` file in place or copy it into the *application_root/module_name/WEB-INF/classes* directory. The new .class file will not trigger a reloading of the application.
- Restart the application. Use the administrative console to restart the application. Or run the `wasadmin stopApplication` and `startApplication` commands. After the application restarts, the new servlet is available for service.

Hot deployment: Yes
Dynamic reloading: Not applicable

- Change the `web.xml` file of a WAR file.
 1. Edit the `web.xml` file in place or copy it into the *metadata_root/module_name/WEB-INF* directory.
 2. Restart the application. Use the administrative console to restart the application. Or run the `wasadmin stopApplication` and `startApplication` commands.

Hot deployment: Yes

Dynamic reloading: Yes

- Change the `ibm-web-ext.xmi` file of a WAR file. Edit the extension settings as needed. You can change all of the extension settings. The only warning is if you set the `reloadInterval` property to zero (0) or the `reloadEnabled` property to false, the application will no longer automatically detect changes to class files. Both of these changes disable the automatic reloading function. The only way to re-enable automatic reloading is to change the appropriate property and restart the application. See other task descriptions in this file for information on restarting an application.

Hot deployment: Not applicable

Dynamic reloading: Yes

- Change the `ibm-web-bnd.xmi` file of a WAR file.
 1. Edit the bindings as needed. You can change all of the values but ensure that the entities you are binding to are present in the configuration of the server.
 2. Restart the application. Use the administrative console to restart the application. Or run the `wasadmin stopApplication` and `startApplication` commands.

Hot deployment: Not applicable

Dynamic reloading: Yes

Changing or adding EJB Jar files

You can change EJB Jar files on application servers without having to stop the server and start it again. This file describes--

- Changing the `ejb-jar.xml` file of an EJB Jar file
- Changing the `ibm-ejb-jar-ext.xmi` or `ibm-ejb-jar-bnd.xmi` file of an EJB Jar file
- Changing the `Table.ddl` file for an EJB Jar file
- Changing the `Map.mapxmi` or `Schema.dbxmi` file for an EJB Jar file
- Updating the implementation class for an EJB file or a dependent class of the implementation class for an EJB file
- Updating the Home/Remote interface class for an EJB file
- Adding a new EJB file to an existing EJB Jar file
- Change the `ejb-jar.xml` file of an EJB Jar file. Restart the application. Automatic reloading will not detect the change. Use the administrative console to restart the application. Or run the `wasadmin stopApplication` and `startApplication` commands.

Hot deployment: Not applicable

Dynamic reloading: Yes

- Change the `ibm-ejb-jar-ext.xmi` or `ibm-ejb-jar-bnd.xmi` file of an EJB Jar file. Restart the application. Automatic reloading will not detect the change. Use the administrative console to restart the application. Or run the `wasadmin stopApplication` and `startApplication` commands.

Hot deployment: Not applicable

Dynamic reloading: Yes

- Change the `Table.ddl` file for an EJB Jar file. Rerun the DDL file on the user database server. Changing the `Table.ddl` file has no effect on the application server and is a change to the database table schema for the EJB files.

Hot deployment: Not applicable
Dynamic reloading: Not applicable

- Change the Map.mapxmi or Schema.dbxmi file for an EJB Jar file.
 1. Change the Map.mapxmi or Schema.dbxmi file for an EJB Jar file.
 2. Regenerate the deployed code artifacts for the EJB file.
 3. Apply the new EJB Jar file to the server.
 4. Restart the application. Use the administrative console to restart the application. Or run the wasadmin stopApplication and startApplication commands.

Hot deployment: Not applicable
Dynamic reloading: Yes

- Update the implementation class for an EJB file or a dependent class of the implementation class for an EJB file.
 1. Update the class file in the *application_root/module_name.jar* file.
 2. If automatic reloading is enabled, you do not need to take further action. Automatic reloading will detect the change. If automatic reloading is not enabled, restart the application of which the EJB file is a member. If the updated module is used by other modules in other applications, restart those applications as well. Use the administrative console to restart the application. Or run the wasadmin stopApplication and startApplication commands.

Hot deployment: Not applicable
Dynamic reloading: Yes

- Update the Home/Remote interface class for an EJB file.
 1. Update the interface class of the EJB file.
 2. Regenerate the deployed code artifacts for the EJB file.
 3. Apply the new EJB Jar file to the server.
 4. If automatic reloading is enabled, you do not need to take further action. Automatic reloading will detect the change. If automatic reloading is not enabled, restart the application of which the EJB file is a member. Use the administrative console to restart the application. Or run the wasadmin stopApplication and startApplication commands.

Hot deployment: Not applicable
Dynamic reloading: Yes

- Add a new EJB file to an existing EJB Jar file.
 1. Apply the new or updated Jar file to the *application_root* location.
 2. If automatic reloading is enabled, you do not need to take further action. Automatic reloading will detect the change. If automatic reloading is not enabled, restart the application. Use the administrative console to restart the application. Or run the wasadmin stopApplication and startApplication commands.

Hot deployment: Yes
Dynamic reloading: Yes

Changing the HTTP plug-in configuration

You can change the HTTP plug-in configuration without having to stop the server and start it again. This file describes--

- Changing the application.xml file to change the context root of a WAR file
- Changing the web.xml file to add, remove, or modify a servlet mapping
- Changing the server.xml file to add, remove, or modify an HTTP transport or changing the virtualhost.xml file to add or remove a virtual host or to add, remove, or modify a virtual host alias

Changing the application.xml file to change the context root of a WAR file

1. Change the application.xml file.
2. Regenerate the plug-in configuration file using the administrative console or by running the GenPluginCfg.bat/sh script.

Hot deployment: Yes
Dynamic reloading: No

Changing the web.xml file to add, remove, or modify a servlet mapping

1. Change the web.xml file.
2. Regenerate the plug-in configuration file using the administrative console or by running the GenPluginCfg.bat/sh script.

If the Web application has file serving enabled or has a servlet mapping of /, you do not have to regenerate the plug-in configuration. In all other cases the regeneration is required.

Hot deployment: Yes
Dynamic reloading: Yes

Changing the server.xml file to add, remove, or modify an HTTP transport or changing the virtualhost.xml file to add or remove a virtual host or to add, remove, or modify a virtual host alias

1. Change the server.xml file to add, remove, or modify an HTTP transport or change the virtualhost.xml file to add or remove a virtual host or to add, remove, or modify a virtual host alias.
2. Regenerate the plug-in configuration file using the administrative console, by running the GenPluginCfg.bat/sh script, or by running a wsadmin command.

Hot deployment: Yes
Dynamic reloading: Yes

Uninstalling applications

After an application no longer is needed, you can uninstall it. Uninstalling an application deletes the application from the WebSphere Application Server configuration repository and it deletes the application binaries from the file system of all nodes where the application modules are installed.

1. Click **Applications > Enterprise Applications** in the administrative console navigation tree to access the Enterprise Applications page.
2. Stop the application. Place a checkmark in the check box beside the application you want uninstalled and click **Stop**. Note that it is recommended you stop the application before uninstalling. However, even if the application is running

when uninstallation commences, the uninstallation program stops the application before deleting application binaries from the file system.

3. Back up the application. Place a checkmark in the check box beside the application you want uninstalled and click **Export** to export the application to an EAR file and preserve the binding information.
4. With a checkmark in the check box beside the application you want uninstalled, click **Uninstall**.
5. Confirm the uninstallation operation.
6. Click **Save** on the console taskbar to save changes made to the administrative configuration.

In the single-server (base) product, application binaries are deleted after you click **Save**. In the Network Deployment product, application binaries are deleted when configuration changes on the deployment manager synchronize with configurations for individual nodes.

Deploying and managing applications: Resources for learning

Use the following links to find relevant supplemental information about deploying and managing applications using the administrative console. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

View links to additional information about:

- Programming model and decisions
- Programming instructions and examples
- Programming instructions and examples

Programming model and decisions

- The J2EE™ Tutorial: The Duke's Bank Application
- Best Practices in WebSphere Application: Separating the developers from the administrators
- Designing Enterprise Applications with the Java™ 2 Platform, Enterprise Edition, Second Edition
- Designing Enterprise Applications, Second Edition
- Building Java™ Enterprise Applications Volume I: Architecture

Programming instructions and examples

- WebSphere Application Server education
- Developing and Testing a Complete 'Hello World' J2EE Application with IBM WebSphere Studio Application Developer for Linux
- Writing Enterprise Applications with Java™ 2 Platform, Enterprise Edition

Administration

- Listing of all IBM WebSphere Application Server Redbooks

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter 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
500 Columbus Avenue
Thornwood, New York 10594 USA

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
522 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Requests

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks and service marks

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

- AIX
- CICS
- Cloudscape
- DB2
- DFSMS
- Everyplace
- iSeries
- IBM
- IMS
- Informix
- iSeries
- Language Environment
- MQSeries
- MVS
- OS/390
- RACF
- Redbooks
- RMF
- SecureWay
- SupportPac
- ViaVoice
- VisualAge
- VTAM
- WebSphere
- z/OS
- zSeries

The term CORBA used throughout this book refers to Common Object Request Broker Architecture standards promulgated by the Object Management Group, Inc.

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

The Duke logo is a trademark or registered trademark of Sun Microsystems, Inc. in the United States and other countries.

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

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

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