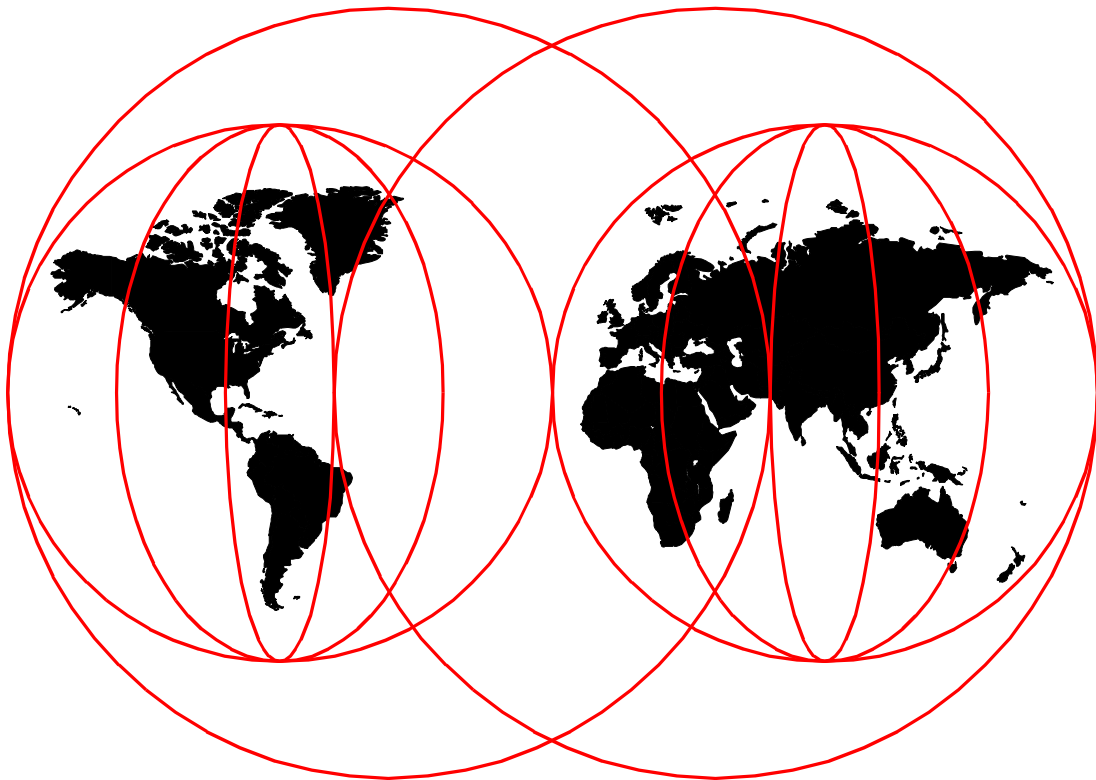


WebSphere Application Servers: Standard and Advanced Editions

Barry Nusbaum, Matias Djunatan, Wakako Jinno, Peter Kelley



International Technical Support Organization

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



International Technical Support Organization

SG24-5460-00

**WebSphere Application Servers:
Standard and Advanced Editions**

July 1999

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix B, "Special Notices" on page 459.

First Edition (July 1999)

This edition applies to V2.02 of WebSphere Application Server for WIndows NT and AIX.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HZ8 Building 678
P.O. Box 12195
Research Triangle Park, NC 27709-2195

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

© Copyright International Business Machines Corporation 1999. All rights reserved.

Note to U.S Government Users - Documentation related to restricted rights - Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Preface	ix
The Team That Wrote This Redbook	ix
Comments Welcome	x
Chapter 1. Planning for the IBM WebSphere Application Server	1
1.1 Overview of WebSphere	1
1.1.1 WebSphere Application Server	2
1.1.2 WebSphere Studio	4
1.1.3 WebSphere Performance Pack	5
1.1.4 WebSphere Site Analysis	5
1.2 The Value of a Web Application Server	5
1.3 Terminology	5
1.3.1 Web Application Servers	5
1.3.2 Servlets	5
1.3.3 Java Server Pages	6
1.3.4 Java Beans	6
1.3.5 Enterprise Java Beans	6
1.3.6 Connectors	6
1.3.7 XML	6
1.3.8 XSL Stylesheets	6
1.3.9 e-business	7
1.3.10 Component Broker	7
1.3.11 Scalability	7
1.3.12 Clustering	7
1.3.13 CORBA	7
1.3.14 RMI	7
1.3.15 IIOP	7
1.3.16 JNDI	7
1.3.17 JDBC	8
1.3.18 Persistence	8
1.3.19 Bean Managed Persistence	8
1.3.20 Container Managed Persistence	8
1.4 Planning for WebSphere Standard Edition	9
1.5 Planning for WebSphere Advanced Edition	10
1.6 Infrastructure Used in This Project	11
1.7 WebSphere Components Overview	11
1.7.1 Static HTML Requests	12
1.7.2 Servlet Requests	13
1.7.3 JSP Requests	18
1.7.4 EJB Interactions	19
Chapter 2. Installation of WebSphere and Associated Products	29
2.1 Infrastructure Installation for Windows NT V4.0	29
2.1.1 JDK 1.1.6	30
2.1.2 HTTP Server V1.3.3	32
2.1.3 Server Modules	34
2.1.4 Domino Go Webserver	37
2.1.5 Netscape SuiteSpot V3.x for Windows NT	41
2.1.6 Microsoft IIS V4.0	47
2.1.7 DB2 Universal Database (UDB) for Windows NT	52
2.1.8 Lotus Domino R5 Server for Windows NT	57

2.2	WebSphere Installation on Windows NT	63
2.3	Infrastructure Installation for AIX V4.3.2	67
2.3.1	JDK 1.1.6 Installation and Setup Procedure	67
2.3.2	IBM HTTP Server V1.3.3 for AIX	70
2.3.3	Domino Go Webserver	72
2.3.4	Netscape SuiteSpot V3.X for AIX	72
2.3.5	DB2 Universal Database (UDB) for AIX	74
2.3.6	Lotus Domino R5 Server for AIX	77
2.4	WebSphere Installation on AIX	79
2.5	Using WebSphere for the First Time	82
2.6	Setting Up a Development System	86
2.6.1	Setting Up VisualAge for Java	87
2.6.2	Setting Up and Using Command Line Session	89
Chapter 3. Content Presentation		91
3.1	How to Deploy and Configure a Servlet	91
3.1.1	Placing Class Files on the Application Server	92
3.1.2	Placing HTML, JSP, and SHMTL Files on the Application Server	94
3.1.3	Configuring a Servlet	96
3.1.4	Monitoring Servlets	116
3.2	Java Server Pages	120
3.2.1	JSP Architecture	120
3.2.2	JSP File Contents	123
3.2.3	<SERVLET> Tags	124
3.2.4	JSP Syntax	127
3.2.5	JSP APIs	140
3.2.6	JSP Sample1	141
3.2.7	JSP Sample 2	145
3.2.8	Tools for Creating JSP Files	152
3.3	Using the WebSphere XML Tools	152
3.3.1	Environment	153
3.3.2	Setting Up the Environment	154
3.3.3	Processing XML	154
3.3.4	XML Catalogs	163
3.3.5	XML Style Sheets and LotusXSL	164
3.3.6	Example: Using XSL and XML to Format DB2 Data	166
3.3.7	Installing Later Versions of the XML Tools	185
Chapter 4. Enterprise Java Services		187
4.1	The EJS Java Processes	187
4.2	Configuring Enterprise Java Services	188
4.2.1	Setting Up the Environment	188
4.2.2	Working with Containers	189
4.2.3	Deploying an EJB	192
4.2.4	Working with Deployment Descriptors Using the Jet Tool	197
4.3	Coding WebSphere EJB Clients	207
4.3.1	Finding EJBs	207
4.3.2	Monitoring EJS	213
4.4	Running the EJS Samples	215
4.4.1	EJS Sample Configuration Steps	216
4.4.2	Running the EJS Samples	217
Chapter 5. Designing Applications for WebSphere		221
5.1	Session Management	221

5.1.1	Maintaining HTTP Sessions	221
5.1.2	Session Tracking in the WebSphere Application Server	225
5.1.3	Session Object Sample	233
5.1.4	Session Clustering	248
5.2	User Profiles	253
5.2.1	Setting Up User Profiles	253
5.2.2	How to Use UserProfile in Your Servlet	255
5.2.3	UserProfile Sample	256
5.2.4	Linking User Profiles to Sessions	265
5.2.5	Extending the UserProfile Class	271
5.3	Using the Personalization Utilities	286
5.3.1	Creating Bulletins	286
5.3.2	Web Site Messaging	295
5.4	Connection Pooling	306
5.4.1	Key Terms	306
5.4.2	Connection Manager Architecture	307
5.4.3	Creating Connection Manager Applications	320
Chapter 6. Enterprise Access		331
6.1	JDBC	332
6.1.1	JDBC Concepts	332
6.1.2	Using JDBC in Java Programs	337
6.1.3	SQLJ	340
6.1.4	Using SQLJ in Java Programs	345
6.2	Using DB2 UDB for WebSphere Applications	349
6.2.1	DB2 Java Support	350
6.2.2	Setting Up DB2 Java Support for the WebSphere Environment	353
6.2.3	DB2 Java Examples	353
6.3	Using Oracle for WebSphere Applications	356
6.3.1	Oracle Java Support	356
6.3.2	Setting Up Oracle Java Support for the WebSphere Environment	359
6.3.3	Oracle Java Examples	359
6.4	Using MQSeries for WebSphere Applications	362
6.4.1	MQSeries Overview	363
6.4.2	MQSeries for Java	365
6.4.3	MQSeries for Java Example	371
6.5	Using TXSeries for WebSphere Application	375
6.5.1	IBM CICS Gateway for Java	376
6.5.2	Setting Up CICS Gateway for Java for WebSphere	381
6.5.3	CICS Gateway for Java Example	383
Chapter 7. WAS 3.0, Site Analyzer Technology Preview		387
7.1	Installing WebSphere Site Analyzer	387
7.1.1	First Time Setup	390
7.2	A First Look at the Site Analyzer	392
7.2.1	Site Analyzer Users	393
7.2.2	Site Analyzer Analysis and Projects	394
7.2.3	Site Analyzer Architecture	394
7.3	Content Analysis	394
7.3.1	Using Content Analysis	396
7.4	Usage Analysis	399
7.4.1	Web Server Log Files	400
7.4.2	Using Usage Analysis	401

7.5	Site Surveyor	404
7.6	Reports and Details	406
7.6.1	Report Element	406
7.6.2	Report	408
Chapter 8. Problem Determination		411
8.1	WebSphere Log Files	411
8.1.1	Overall Log Structure	411
8.1.2	The JVM Standard Error Log	412
8.1.3	The JVM Standard Output Log	413
8.1.4	The IBM HTTP Server Information Log	413
8.1.5	The IBM HTTP Server Error Log	414
8.1.6	The WebSphere Trace Log	414
8.1.7	The Servlet Admin Service Error Log	415
8.1.8	The Servlet Admin Service Event Log	416
8.1.9	The Servlet Admin Service Access Log	416
8.1.10	The Servlet Service Error Log	417
8.1.11	The Servlet Service Event Log	418
8.1.12	The Servlet Service Access Log	418
8.1.13	The WebSphere Engine Tracing Log	418
8.2	The Application Server Debug Console	418
8.2.1	Enabling the Console	418
8.2.2	The Server Console Monitor	419
8.2.3	The Trace Enabler Page	420
8.2.4	The Exceptions Monitor	421
8.2.5	The EJS Status Monitor	422
8.2.6	The Resource Usage Monitor	422
8.2.7	The Loaded Servlets Monitor	423
8.2.8	The Sessions Monitor	424
8.2.9	The Pooled Connections Monitor	424
8.3	Tracing	424
8.3.1	Tracers	425
8.3.2	Trace Output Handlers	427
8.3.3	Running the Socket Server Trace Console	427
8.3.4	Setting Trace Properties Using the Debug.properties File	428
8.3.5	Creating Your Own Tracers	430
8.3.6	Creating Your Own Trace Output Handlers	431
8.4	The Server Execution Analysis Pages	435
8.4.1	The JVM Debug Page	436
8.4.2	The Settings Pane	436
8.4.3	The Error Log Settings Page	438
8.4.4	The Event Log Settings Page	439
8.4.5	The Dump Panel Setup Page	440
8.4.6	The Log Output Monitor Page	441
8.5	Miscellaneous Debugging Tools	442
8.5.1	DB2 CLI Tracing	442
8.5.2	JDBC Output Redirection	444
8.5.3	Running the EJS Processes Stand-alone	445
Appendix A. WebSphere Samples		449
A.1	Other Configuration Steps	455
A.1.1	Samples From the Samples Web Page	456

Appendix B. Special Notices	459
Appendix C. Related Publications	461
C.1 International Technical Support Organization Publications	461
C.2 Redbooks on CD-ROMs	461
C.3 Other Publications	461
C.4 Web Sites Referenced in This Book	461
How to Get ITSO Redbooks	465
IBM Redbook Fax Order Form	466
Index	467
ITSO Redbook Evaluation	469

Preface

IBM WebSphere is a collection of software products that work on IBM and non-IBM platforms to help you develop and manage Web sites. The WebSphere Application Servers help provide the infrastructure for deploying your Web applications.

In this book we show how to plan for, install and use the WebSphere Application Servers on the AIX and NT platforms. Explanations are given for common Internet-related terms with pointers to examples of how to implement many of them within the WebSphere structure. Details are provided on how to build applications and connect to back-end DB2 and Oracle databases. In addition to building applications we take a first look at how to manage Web sites and how to perform problem determination.

This book can be used by webmasters and HTML coders to help efficiently design their application infrastructure. In addition, anyone building a new WebSphere environment from scratch will greatly benefit from the step-by-step approach that is shown in the installation chapter.

The Team That Wrote This Redbook

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

Barry Nusbaum is a Consulting International Technical support representative at the International Technical Support Organization, Raleigh Center. He writes extensively and teaches IBM classes worldwide on all areas of systems management on the NT and AIX platform. He is also currently working on projects related to the e-business Application Framework and WebSphere. Before joining the ITSO seven years ago, he worked in Professional Services in the United States as a National Communications Specialist. You can reach him by e-mail at bnusbaum@us.ibm.com.

Matias Djunatan is a solution architect in PT. Mitra Integrasi Komputindo (MIK), Indonesia. He has eight years of experience in developing applications on various IBM platforms. He holds a degree in Computer Science from the University of New South Wales, Australia. His areas of expertise include finance/banking information technology, middleware, distributed component architecture and Java application developments.

Wakako Jinno is an I/T engineer in Japan. She has experience in designing and developing Web applications associated with RDB, with Java, embedded SQL, etc. She has worked at Internet Systems, IBM Japan Systems Engineering for two years. Her areas of expertise include WebSphere, RDB such as DB2, Oracle, Sybase.

Peter Kelley is an advisory technical specialist at the Sydney Solution Partnership Centre in Australia. He has 10 years of experience working with IBM software technologies, including three years with Java. He holds a Bachelor of Science degree from the University of Sydney. His areas of expertise include Java, VisualAge for Java and WebSphere. He has completed one previous redbook on the OS/2 Workplace Shell.

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

Scott Boag
Lotus Development Corp.

Elias Bayeh, Arnold Goldberg, Jason McGee, Michael Morton, Spike Washburn
IBM WebSphere Development Team

Chris Beckett, Mark Fisher, Tom Hartrick, Ken McCauley, Lisa Morley, Jeff Reser
IBM WebSphere Development Team

Shawn Walsh
International Technical Support Organization, Raleigh Center

Comments Welcome

Your comments are important to us!

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

- Fax the evaluation form found in "ITSO Redbook Evaluation" on page 469 to the fax number shown on the form.
- Use the online evaluation form found at <http://www.redbooks.ibm.com/>
- Send your comments in an Internet note to redbook@us.ibm.com

Chapter 1. Planning for the IBM WebSphere Application Server

In this book the following conventions apply:

- Directory paths on AIX and Windows NT have slashes that go in different directions. In all cases when you see slashes going the opposite way from the platform that you are working on they can be reversed.

For example

```
<Server Root>\servlets\WebBank\CreateAccountServlet.servlet
```

in this book for Windows NT is the same as

```
<Server Root>/servlets/WebBank/CreateAccountServlet.servlet
```

when you are working on AIX, and vice versa.

- The symbol <Server Root> refers to the root directory where the IBM WebSphere Application Server is installed. By default this will be C:\WebSphere\AppServer on Windows NT and /usr/WebSphere/AppServer on AIX, but your installation may vary.
- The symbol <your web server> refers to the Web address of the machine on which you have the WebSphere Application Server installed. This may be as simple as localhost if the machine you are working on is the local machine or something like server.your.company.com if WebSphere is installed on another (remote) machine.

1.1 Overview of WebSphere

The IBM WebSphere products are a group of products designed to assist Web application developers to develop, deploy and manage advanced Web sites. The WebSphere product range includes the following:

- WebSphere Application Server for the deployment of Web applications (the subject of this book). WebSphere Application Server V2.02 currently comes in both a Standard and an Advanced Edition. See 1.1.1, “WebSphere Application Server” on page 2 for more details.
- WebSphere Studio for the development of Web applications. See 1.1.2, “WebSphere Studio” on page 4 for more details.
- WebSphere Performance Pack for enhancing and managing the performance of WebSphere Web sites. See 1.1.3, “WebSphere Performance Pack” on page 5 for more details.
- WebSphere Site Analysis for monitoring client requests to a WebSphere Web site and analyzing links on that site (see Chapter 7, “WAS 3.0, Site Analyzer Technology Preview” on page 387 for more details). WebSphere Site Analysis is now included as part of WebSphere Application Server Version 3.0 and not as a separate product. It is included in this book as a technology preview. That means that some of the screens and functions will change when the product is generally available.

Although we refer to the other components in the WebSphere product line we primarily are concerned with the WebSphere Application Server and to some extent with WebSphere Site Analysis. The terms WebSphere and WebSphere Application Server (or WAS) will be used interchangeably even though

WebSphere refers to the entire product line. Other products will be referred to by their full names.

1.1.1 WebSphere Application Server

The WebSphere Application Server (WAS) is a Java application server designed to facilitate the management and deployment of Web applications. These deployed applications are typically composed of either Enterprise Java Beans (EJBs), Java Server Pages (JSPs) and/or Java servlets and they communicate to clients using a Web browser client interface. Each of the different types of Java applications that run in the WebSphere environment can also make use of Java Beans (which are different from Enterprise Java Beans). WebSphere provides the environment and infrastructure required to install and manage these types of applications.

WAS does not operate in isolation. It needs to be installed on a host Web server that handles HTTP requests from browsers and delivers HTML back to them using the HTTP protocol. When WebSphere is installed, it modifies the configuration of its host Web server to redirect certain requests to WebSphere for processing rather than letting the Web server handle them. WebSphere can be installed on a number of supported Web servers (see 1.5, "Planning for WebSphere Advanced Edition" on page 10) and it also ships with the IBM HTTP Server, which is based on the popular Apache Web server but adds SSL support.

The WebSphere application server processes can either run as part of the Web Server processes or separately, using interprocess communication to talk to the Web server.

WebSphere makes use of a Java development and run-time environment on the host machine. This Java environment allows WebSphere to execute the Java programs that make up the Web applications.

WebSphere is administered through the use of a Java-capable Web browser that supports HTML V4, such as Netscape Navigator or Microsoft Internet Explorer (see Table 1., "AIX Requirements" on page 9 for supported browsers). This is done by using an HTTP administration interface that allows remote administration of the server. The administration interface can be accessed by loading the URL `http://<your web server>:9527/` into a Web browser. If your Web browser is on the same system as your server you can also perform the administration.

WebSphere Advanced uses a database (see 1.5, "Planning for WebSphere Advanced Edition" on page 10) to provide persistence (see 1.3.18, "Persistence" on page 8) services for storing EJBs. If a database is not installed or available on the machine on which WebSphere is installed, then DB2 UDB V5.2 is installed during the WAS installation for use with WAS only. A limited function version of DB2 is packaged with WebSphere on the product CD.

WebSphere Application Server V2.02 comes in two editions: Standard and Advanced. The Standard Edition includes support for Java Server Pages (JSPs) and servlets as well as XML document structure services and session management. The Advanced Edition has all of the same functions as the Standard Edition, plus it has an EJB engine and a database to act as a persistent store for EJB information. A third edition, called Enterprise Edition, will be available later in 1999 when WebSphere V3.0 is delivered.

The features in WebSphere Application Server Standard Edition are:

- IBM HTTP Server

WebSphere Application Server ships with IBM HTTP server, which can be installed as part of the WebSphere Application Server installation process if no other Web server is installed (See Chapter 2, “Installation of WebSphere and Associated Products” on page 29 for more information on WebSphere installation). The IBM HTTP Server is based on the Apache freeware Web server with added SSL support.

- Servlet support

WebSphere includes a servlet engine for running Java Servlets. See 3.1, “How to Deploy and Configure a Servlet” on page 91 for more information on servlets.

- Support for Lotus Domino Version 5.0

See 2.1.8, “Lotus Domino R5 Server for Windows NT” on page 57 for more information on installing WebSphere with Lotus Domino Version 5.0.

- Administration interface

WebSphere is administered using a Web-based Java administration interface.

- XML Document Structure Services

WebSphere provides XML Document Structure Services to allow the generation and manipulation of XML-formatted data using Java. See 3.3, “Using the WebSphere XML Tools” on page 152 for more information on working with XML Document Structure Services.

- Integration with VisualAge for Java

IBM VisualAge for Java includes a WebSphere test environment to allow the development and testing of WebSphere applications within the VA Java IDE. See

<http://www7.software.ibm.com/vad.nsf/Data/Document3172?OpenDocument&SubMast=1>

for more information on integrating VisualAge for Java and WebSphere.

1.1.1.1 Features in WebSphere Application Server Advanced Edition

The following additional features are present in the Advanced Edition:

- Enterprise JavaBeans Server allows the deployment of EJBs

EJBs provide a platform-independent way to provide business logic in a managed environment.

- A naming service accessible through JNDI to facilitate the location and usage of EJBs by remote clients

A naming service is used by EJB clients to locate references to EJBs deployed on the WebSphere server.

- CORBA support to allow EJBs deployed in WebSphere to be accessed by remote CORBA objects

CORBA support facilitates interoperability with CORBA objects on non-EJB servers.

These features are together called Enterprise Java Services (EJS) and are described in Chapter 4, “Enterprise Java Services” on page 187.

1.1.2 WebSphere Studio

WebSphere Studio is a collection of development tools that are used to develop the components necessary to produce a Web site. WebSphere Studio includes IBM VisualAge for Java Professional Edition, Net Objects Fusion, Net Objects Bean Builder, Net Objects Script Builder and the WebSphere Studio Workbench and Wizards. Version 3.0 will include the Wallop Build-IT product as well.

1.1.2.1 IBM VisualAge for Java

IBM VisualAge for Java, or VA Java, is an integrated enterprise Java development environment. It is a repository-based tool that allows incremental compilation of Java source as well as powerful version control features. It includes a visual programming environment for Java component assembly and many features that assist the development of enterprise Java programs. It provides strong support for databases and other back-end systems as well as Java Beans and Enterprise Java Beans. It comes in Entry, Professional and Enterprise versions. For more information see <http://www.software.ibm.com/ad/vajava>.

For additional details and examples about VA Java go to <http://www.redbooks.ibm.com> and download the following books:

- *VisualAge for Java Enterprise Version 2: Data Access Beans - Servlets - CICS Connector*, SG24-5265
- *Using VisualAge for Java Enterprise Edition Version 2 to Develop CORBA EJB Applications*, SG24-5276
- *Programming with VisualAge for Java Version 2*, SG24-5264

1.1.2.2 Net Objects Fusion

Net Objects Fusion is a tool that allows the creation of entire Web sites using a template-based approach. Each template captures a particular look and feel with facilities to add different Web site components such as text, graphics and plug-ins to each page as it is created. For more information see <http://www.netobjects.com/products/html/nf3i.html>.

1.1.2.3 Net Objects Bean Builder

Net Objects Bean Builder is a Java Bean assembly tool. It allows you to visually assemble Java Beans and link their behavior. The finished code can then be packaged as a Java Bean or applet as well as to a variety of other formats. For more information see <http://www.netobjects.com/products/html/nbb1.html>.

1.1.2.4 Net Objects Script Builder

Net Objects Script Builder is an integrated development environment for developing scripting for Web pages. It includes support for the scripting technologies used in Web servers from IBM, Sun, Microsoft and Netscape. The IDE supports syntax highlighting for scripting languages and there is an integrated debugger. For more information see <http://www.netobjects.com/products/html/nsb3.html>.

1.1.2.5 WebSphere Studio Workbench and Wizards

The WebSphere Studio Workbench is a workbench that groups together all of the files that make up a Web site under development. Each file, or group of files, can be edited with any of the Studio development tools and new tools can be added. The wizards facility allows the easy creation of Java servlets (see 1.3.2,

“Servlets” on page 5) to access databases, use server-side Java Beans and capture information about site visitors. For more information see <http://www.software.ibm.com/webserverstudio/doc/wsguide.pdf>.

1.1.3 WebSphere Performance Pack

For more information, see <http://www.redbooks.ibm.com> and download *IBM WebSphere Performance Pack Usage and Administration*, SG24-5233.

1.1.4 WebSphere Site Analysis

IBM WebSphere Site Analysis is a tool used to provide basic traffic measurement functions for a Web site. It allows users to gauge traffic volume (hits, visits), identify traffic sources (domains, subdomains, referrers), and manage site integrity (link verification, site conformance). For more information see <http://www.software.ibm.com/webserveranalysis> or Chapter 7, “WAS 3.0, Site Analyzer Technology Preview” on page 387.

1.2 The Value of a Web Application Server

A Web application server, such as WebSphere, is a key component in a three-tier (or n-tier) e-business solution. It acts as an integration point between the enterprise data and applications on the back end and a nearly universal client, the browser, on the front end. This allows both a low-cost client platform with low configuration overhead and access for a wide variety of client devices without changing the application. It also shields the back-end servers from interference generated by client requests and improves application scalability by allowing resource pooling at the middle tier.

A complete discussion of the architecture and thinking behind three-tier computing is probably beyond the scope of this book. However, there is a good set of white papers on the subject at <http://www.software.ibm.com/ebusiness/library.html>, and in particular the architecture overview white paper at http://www.software.ibm.com/ebusiness/arch_overview.html is very interesting.

1.3 Terminology

The following are some terms used in this book.

1.3.1 Web Application Servers

A Web application server is a software program designed to manage applications at the *second-tier* of three-tier computing, that is, the business logic components. A Web application server manages applications that use data from back-end systems, such as databases and transaction systems, and provides output to a Web browser on a client. For more information see <http://www.software.ibm.com/ebusiness/appsrsvw.html>.

1.3.2 Servlets

Servlets are Java classes that run on Web servers to provide dynamic HTML content to clients. They take as input the HTTP request from the client and output dynamically generated HTML. For more information on servlets see 3.1, “How to

Deploy and Configure a Servlet” on page 91 and also <http://www.software.ibm.com/ebusiness/pm.html#Servlets>.

1.3.3 Java Server Pages

Java Server Pages are HTML source files that include Java extensions to provide dynamic content and increased functionality. Java Server Pages are compiled into Servlets before deployment. See

<http://www.software.ibm.com/ebusiness/pm.html#Java Server Pages> and 3.2, “Java Server Pages” on page 120 for more information.

1.3.4 Java Beans

Java Beans are Java components designed to be used on client systems. They are Java classes that conform to certain coding standards. They can be described in terms of their properties, methods and events. Java Beans may be packaged with a special descriptor class called a BeanInfo class and special property editor classes in a JAR file. Java Beans may or may not be visual components. See <http://www.javasoft.com/beans/docs> for more information.

1.3.5 Enterprise Java Beans

Despite the name, Enterprise Java Beans or EJBs are not Java Beans. Enterprise Java Beans are server-side Java components that are designed for distributed environments. They do not exist in isolation but are rather deployed in containers that provide services such as security, naming and directory services and persistent storage. WebSphere Application Server is just such a container. See <http://java.sun.com/products/ejb/> for more information.

1.3.6 Connectors

The term connectors, or e-business connectors, is used to describe gateway products from IBM that allow access to enterprise data on back-end systems over the Internet. They include direct browser access to back-end systems such as DB2 through Net.data and also Java access through products such as the CICS gateway for Java. See <http://www.software.ibm.com/ebusiness/connectors.html> for more information.

1.3.7 XML

XML, or eXtensible Markup Language, is a platform-independent and application-independent way of describing data using tags. XML (a subset of SGML) is similar to HTML in that it uses tags to describe document elements but different in that the tags describe the structure of the data rather than how the data is to be presented to a client. XML has the facility to allow data providers to define new tags as needed to better describe the data domain being represented. For more information see <http://www.software.ibm.com/xml> and 3.3, “Using the WebSphere XML Tools” on page 152.

1.3.8 XSL Stylesheets

XSL stylesheets are documents that describe a mapping between XML documents and visual data that can be presented to a client in a browser. XSL was a draft standard when this book was being written. The draft can be found at <http://www.w3.org/TR/WD-xs>. An example of using XSL stylesheets with

WebSphere can be found in 3.3.6.4, “Developing an XSL Style Sheet” on page 169.

1.3.9 e-business

e-business is a term used by IBM to describe the use of Internet technologies to transform business processes. What this means in practice is using Internet clients such as Web browsers as front ends for applications that access back-end legacy systems to allow greater access. See

<http://www.software.ibm.com/ebusiness> for more information.

1.3.10 Component Broker

Component Broker is an IBM CORBA management server product that provides an object request broker (ORB) to facilitate the deployment of CORBA objects. The EJB deployment engine in WebSphere is based largely on similar services in Component Broker. See <http://www.software.ibm.com/ad/cb> for more information.

1.3.11 Scalability

Scalability is an abstract attribute of software that refers to its ability to handle increased data throughput without modification. WebSphere handles scalability by allowing execution on a variety of hardware platforms that allow increased performance and clustering.

1.3.12 Clustering

Clustering is a technique used to provide scalability through the use of multiple copies of an application on the same or separate machines. Careful management of the different applications is necessary to ensure that they work together effectively. WebSphere has limited clustering support in Version 2.x and more support in Version 3.0.

1.3.13 CORBA

Common Object Request Broker Architecture (CORBA) is a cross-platform, industry-standard distributed object protocol. CORBA is used to locate and use objects on a variety of platforms, written in a variety of languages across a network. See <http://www.omg.org> for more information on CORBA.

1.3.14 RMI

Remote Method Invocation (RMI) is a lightweight distributed object protocol that allows Java objects to call each other across a network. RMI is part of the core Java specification. See <http://java.sun.com/products/jdk/rmi/index.html> for more information.

1.3.15 IIOP

Internet Inter ORB Protocol (IIOP) is an internet protocol used for CORBA object communication. For more information see <http://www.what-is.com/iiop.htm>.

1.3.16 JNDI

Java Naming and Directory Interface (JNDI) is an API that allows Java programs to interface and query naming and directory services in order to find information about network resources. JNDI is used in WebSphere to provide a directory of

Enterprise Java Beans. See <http://java.sun.com/products/jndi/index.html> for more information.

1.3.17 JDBC

JDBC is a Java API that allows Java programs to communicate with different database management systems in a platform-independent manner. Database vendors provide JDBC drivers for their platforms that implement the API for their database, allowing the Java developer to write applications to a consistent API no matter which database is used. For more information see 6.1, “JDBC” on page 332.

1.3.18 Persistence

Persistence is a term used to describe the storage of objects in a database to allow them to persist over time rather than being destroyed when the application containing them terminates. Enterprise Java Bean containers such as WebSphere provide persistence services for EJBs deployed within them. For more information see 1.3.19, “Bean Managed Persistence” and 1.3.20, “Container Managed Persistence”.

1.3.19 Bean Managed Persistence

Bean Managed Persistence (BMP) is a term used to describe a type of entity EJB where the bean developer specifies how the bean is to be persisted to a database by writing Java code in the appropriate methods to perform the tasks required. See 1.3.20, “Container Managed Persistence”.

1.3.20 Container Managed Persistence

Container Managed Persistence (CMP) is a term used to describe a type of entity EJB where the code to persist the bean to a database is generated at deployment time by the EJB container. See 1.3.19, “Bean Managed Persistence”.

1.4 Planning for WebSphere Standard Edition

The hardware and software requirements for IBM WebSphere Application Server Standard Edition on AIX and Windows NT are as follows.

Table 1. AIX Requirements

Operating System	IBM AIX Version 4.2.1 or Higher
Supported Web Server	IBM HTTP Server V1.3.3 (on WAS installation CD) Apache Server V1.3.2 for AIX Domino V5.0 for AIX Lotus Domino Go Webserver V4.6.2.5 for AIX Netscape Enterprise Server V3.01 and V3.51 for AIX (recommend V3.5.1) Netscape FastTrack Server V3.01 for AIX
JDK	JDK 1.1.6 with patch PTF2 or higher
Supported Web Browsers	A browser that supports JDK1.1: Netscape Navigator 4.06 or higher Microsoft Internet Explorer 4.01 with the fix pack, or higher Sun HotJava 1.1 or higher

- RS/6000 or RS/6000 SP running AIX V4.2.1, or later
- Support for an appropriate network interface
- Minimum 45 MB of free disk space for installation
- CD-ROM drive
- Minimum 64 MB of memory, 128 MB recommended

Table 2. Windows NT Requirements

Operating System	Windows NT Server with Service Pack 3
Supported Web Server	IBM HTTP Server V1.3.3 for NT (available during Application Server install) Apache Server V1.3.2 for NT Domino V5.0 for Windows NT Lotus Domino Go Webserver V4.6.2.5 for NT Microsoft IIS V3.x and V4.0 for NT Netscape Enterprise Server V3.01 and V3.51 for NT (recommend V3.5.1) Netscape FastTrack Server V3.01 for NT
JDK	JDK 1.1.6 (JDK 1.1.7 from IBM or Sun may also work)
Supported Web Browsers	A browser that supports JDK1.1: Netscape Navigator 4.06 or higher Microsoft Internet Explorer 4.01 with the fix pack, or higher. Sun HotJava 1.1 or higher

The software requirements are:

- Any Intel-based PC running Windows NT Server V4.0
- Support for a communications adapter
- Minimum 40 MB of free disk space for installation
- CD-ROM drive
- Minimum 64 MB of memory, 128 MB recommended

1.5 Planning for WebSphere Advanced Edition

The hardware and software AIX requirements for IBM WebSphere Application Server Advanced Edition are as follows:

Table 3. AIX Requirements

Operating System	IBM AIX Version 4.2.1 or Higher
Supported Web Server	IBM HTTP Server V1.3.3 (on WAS installation CD) Apache Server V1.3.2 for AIX Domino V5.0 for AIX Lotus Domino Go Webserver V4.6.2.5 for AIX Netscape Enterprise Server V3.01 and V3.51 for AIX (recommend V3.5.1)
JDK	JDK 1.1.6 with patch PTF2 or higher
Supported Web Browsers	A browser that supports JDK1.1: Netscape Navigator 4.06 or higher Microsoft Internet Explorer 4.01 with the fix pack, or higher Sun HotJava 1.1 or higher
Supported Databases	Any database supporting JDBC Version 1.1

- RS/6000 or RS/6000 SP running AIX V4.2.1, or later
- Support for an appropriate network interface
- Minimum 45 MB of free disk space for installation; 128 MB recommended, especially if installing DB2
- CD-ROM drive
- Minimum 128 MB of memory; 256 MB recommended

Table 4. Windows NT Requirements

Operating System	Windows NT Server with Service Pack 3
Supported Web Server	IBM HTTP Server V1.3.3 for NT (available during Application Server install) Apache Server V1.3.2 for NT Domino V5.0 for Windows NT Lotus Domino Go Webserver V4.6.2.5 for NT Microsoft IIS V3.x and V4.0 for NT Netscape Enterprise Server V3.01 and V3.51 for NT (recommend V3.5.1)
JDK	JDK 1.1.6 (JDK 1.1.7 from IBM or Sun may also work)
Supported Web Browsers	A browser that supports JDK1.1: Netscape Navigator 4.06 or higher Microsoft Internet Explorer 4.01 with the fix pack, or higher Sun HotJava 1.1 or higher
Supported Databases	DB2 UDB 5.2 or higher DB2 UDB 5.0 with FixPak US9077 may work but is not supported

The Windows NT hardware requirements are:

- Support for a communications adapter
- Minimum 40 MB of free disk space for installation; 128 MB recommended, especially if installing DB2
- CD-ROM drive
- Minimum 128 MB of memory; 256 MB recommended

1.6 Infrastructure Used in This Project

The following is a list of hardware and software that was used for this project:

RS/6000:

- RS/6000 43P-140 (7043-140)
- Single 332 MHz PowerPC_604 Processor
- 256 MB RAM
- 2 * 4.5GB SCSI HDD

Intel:

- Netfinity 3000 (8476-21U)
- Single 350 MHz Pentium II Processor
- 1 * 4GB SCSI HDD
- 128 MB RAM

AIX Software:

- AIX 4.3.2
- WebSphere Application Server Standard V2.02
- WebSphere Application Server Advanced V2.02

1.7 WebSphere Components Overview

Figure 1 on page 12 is a logical diagram showing the various flows in an operating WebSphere system. Although there is a lot of information contained here, the data flows that we are concerned with can be broken down into several categories:

- Static HTML requests - See 1.7.1, "Static HTML Requests" on page 12.
- Servlet requests - See 1.7.2, "Servlet Requests" on page 13.
- JSP page requests - See 1.7.3, "JSP Requests" on page 18.
- EJB interactions - See 1.7.4, "EJB Interactions" on page 19.

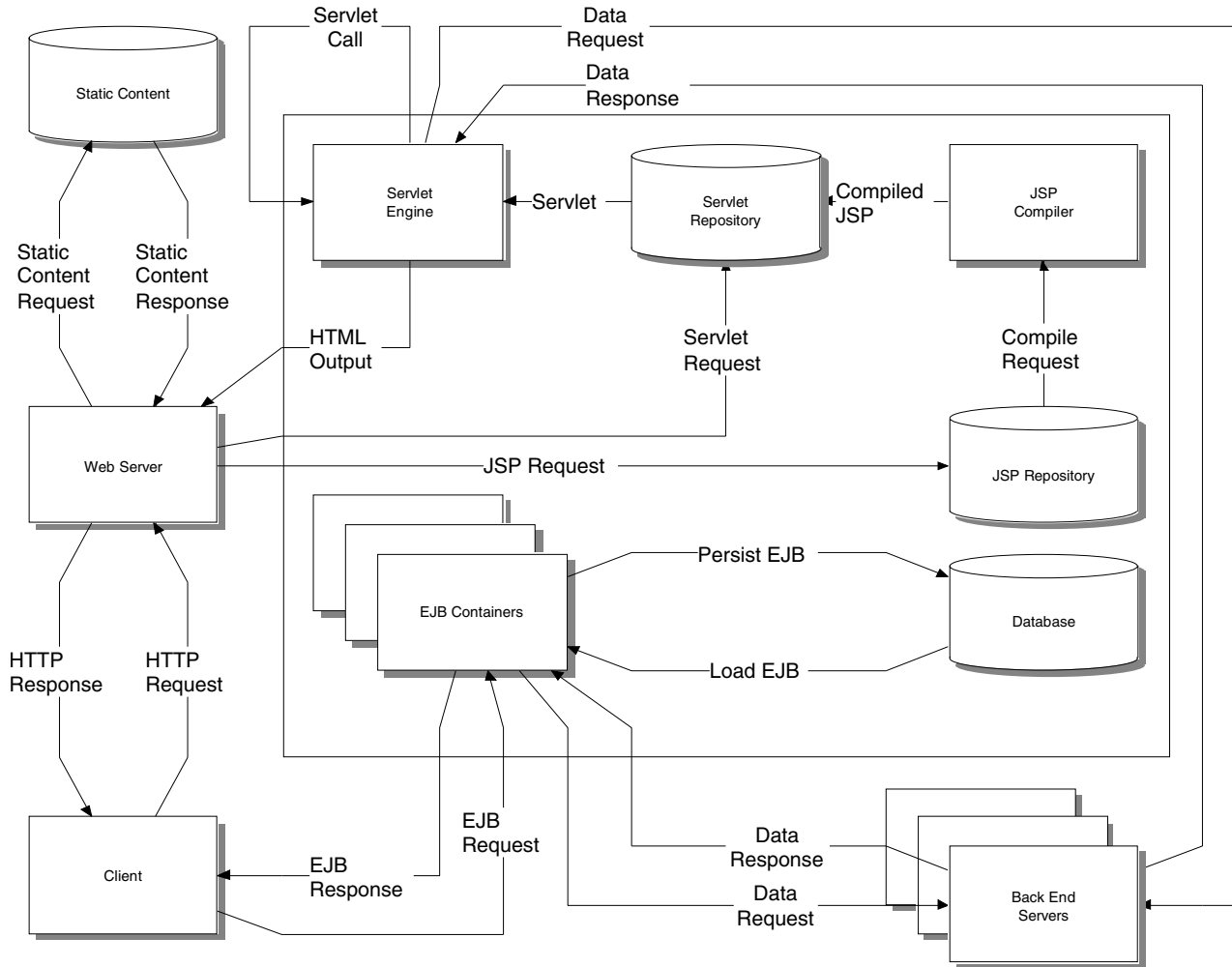


Figure 1. WebSphere Data Flows

1.7.1 Static HTML Requests

Serving static HTML is perhaps the simplest task that a Web server can perform and it does not use the capabilities of WebSphere at all. Instead, it relies entirely on the host Web server. Figure 2 on page 13 shows the flow of data that occurs in servicing a static HTML page request. Upon receiving a request from the client, the Web server retrieves the correct document from the server's file system and sends it to the client. There may be minimal work done by the Web server to translate the name of the document requested by the client into the correct name of the document on the file system.

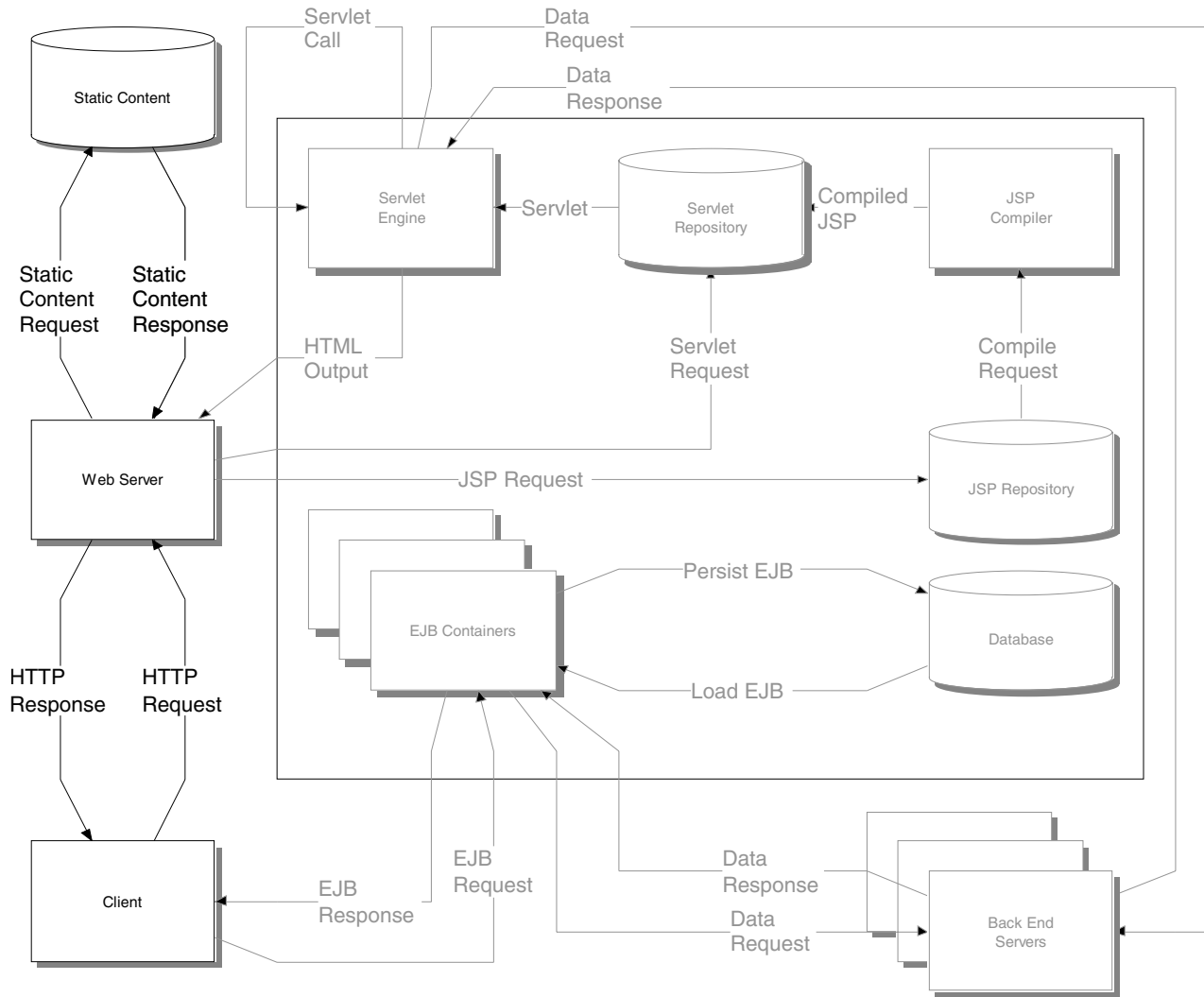


Figure 2. Static HTML Request Data Flows

Although WebSphere does not get involved in this process it is important to realize that most Web sites will have some static HTML pages included alongside the dynamic content provided by WebSphere. The WebSphere Studio tools allow you to generate this static content, and the static components of the more dynamic Web pages (such as those created using JSPs), with a consistent look and feel so that the end user does not notice a difference in the pages. Net Objects Fusion is typically used to perform this task.

1.7.2 Servlet Requests

Servlets provide the base level of WebSphere functions. A servlet is a Java program that produces HTML dynamically for a given client request. The client request is passed as a parameter to the servlet from the application server along with a parameter containing the data structure to be used in constructing the servlet response.

Figure 3 shows the data flows possible for servlet calls.

WebSphere EJS Server. This is why there are no lines linking the servlet engine and the EJB containers in Figure 3; the client interfaces described in 1.3.5, “Enterprise Java Beans” on page 6 are used.

Figure 4 on page 16 shows the Java program listing for a simple servlet. The important points to note are the parameters passed to the servlet of type `HttpRequest` (the variable named `request`) and `HttpResponse` (the variable named `response`). The request is queried through `request.getParameterNames()` and `request.getParameter(key)` to find if a `Name` parameter was specified on the HTTP request. If not, the name is set to “Hello”. The next few lines set the MIME type of the response to “text/html” for output to a browser and some other fields for the response. Next, the program queries the `HttpResponse` to get its output stream (the `PrintWriter out`) and writes HTML back to the client.

```

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Step 1: Extend HttpServlet.
public class ServletSample extends HttpServlet {

    // Step 2: Specify the required methods.
    public void doGet (HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException    {

        // Step 3: Get the HTTP request information, if any.
        Enumeration keys;
        String key;
        String myName = "";
        keys = request.getParameterNames();
        while (keys.hasMoreElements())
        {
            key = (String) keys.nextElement();
            if (key.equalsIgnoreCase("myName"))
                myName = request.getParameter(key);
        }
        System.out.println("Name = ");
        if (myName == "")
            myName = "Hello";
        // Step 4: Create the HTTP response.
        response.setContentType("text/html");
        response.setHeader("Pragma", "No-cache");
        response.setDateHeader("Expires", 0);
        response.setHeader("Cache-Control", "no-cache");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>Just a basic servlet</title></head>");
        out.println("<body>");
        out.println("<h1>Just a basic servlet</h1>");
        out.println ("<p>" + myName + ", this is a very basic servlet that
writes an HTML page. The source code for more interesting sample servlets is
in the Application Server samples/ directory.");
        out.println ("<p>For instructions on running those samples on your
Application Server, open the page:");

        out.println("<pre>http://<em>your.server.name</em>/IBMWebAs/samples/index.
html</pre>");
        out.println("where <em>your.server.name</em> is the hostname of your
Application Server.");
        out.println("</body></html>");
        out.flush();
    }
}

```

Figure 4. ServletSample.java

1.7.2.1 Servlet Filtering and Chaining

If multiple servlets are needed to produce a response to a particular client request then the normal procedure for producing HTML responses becomes a

little more complex. There are two ways for multiple servlets to collaborate on the response: filtering and chaining.

In servlet filtering the servlet changes the MIME type of the response it sends from text/html (which would be sent by WebSphere straight back to the client) to a user-defined MIME type. WebSphere is then configured to associate the user-defined MIME type with a particular servlet and the output of the first servlet is used as the input to the second servlet. In this way servlets can filter the input to other servlets, forming a filter chain if necessary.



Figure 5. Servlet Filtering

In servlet chaining, multiple servlets are called for a single client HTTP request, each servlet providing part of the HTML output. Each servlet receives the original client HTTP request as input and each servlet produces its own output independently. This is accomplished by defining a servlet alias that is specified on the original request and specifying multiple servlets as the target. Each servlet is called in the order specified on the alias and the output HTML is made up of the output from all of the servlets.

Note: This is different from the way that other Web servers perform servlet chaining. Other servers use the output of the first servlet in the chain as the input request to the second servlet in the chain and so on. To achieve this functionality with WebSphere, servlet filtering must be used.

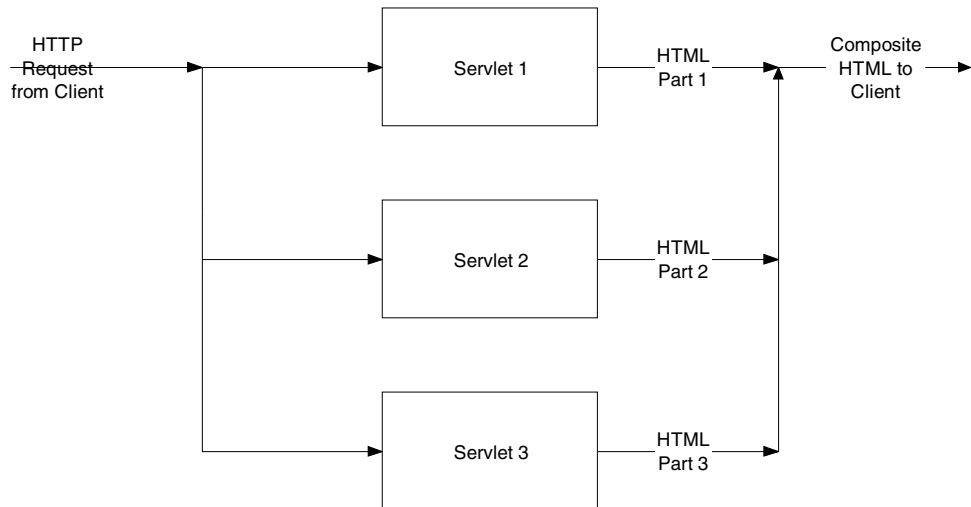


Figure 6. Servlet Chaining

Servlet filtering and chaining have the advantage of allowing the Web developer to create modular servlets that can, for example, output standard HTML headers and footers or provide common dynamic content for pages.

1.7.3 JSP Requests

Java Server Pages (JSPs) are a way for developers who are familiar with HTML to easily create servlets. They are also useful for applications where servlets and other generators of dynamic HTML content must be integrated with static HTML. JSPs, although quite different from servlets at development time, are actually precompiled into servlets at run time. Each JSP is stored in the Web server's normal document hierarchy and the first time it is invoked by a client it is compiled to a servlet by the JSP compiler. It is then stored in the servlet repository (JSP-generated servlets are stored under the directory <Server Root>/servlets/pagecompile) and treated as a servlet for the rest of its life. If the JSP is changed at any time, WebSphere detects the change and recompiles the JSP into an updated version of the servlet, and then runs the new servlet for the client.

Figure 7 shows the data flows for JSP requests in WebSphere. Note that the "JSP Repository" shown in the diagram is a logical entity only. JSP source files are stored in the Web server document hierarchy just like static HTML files.

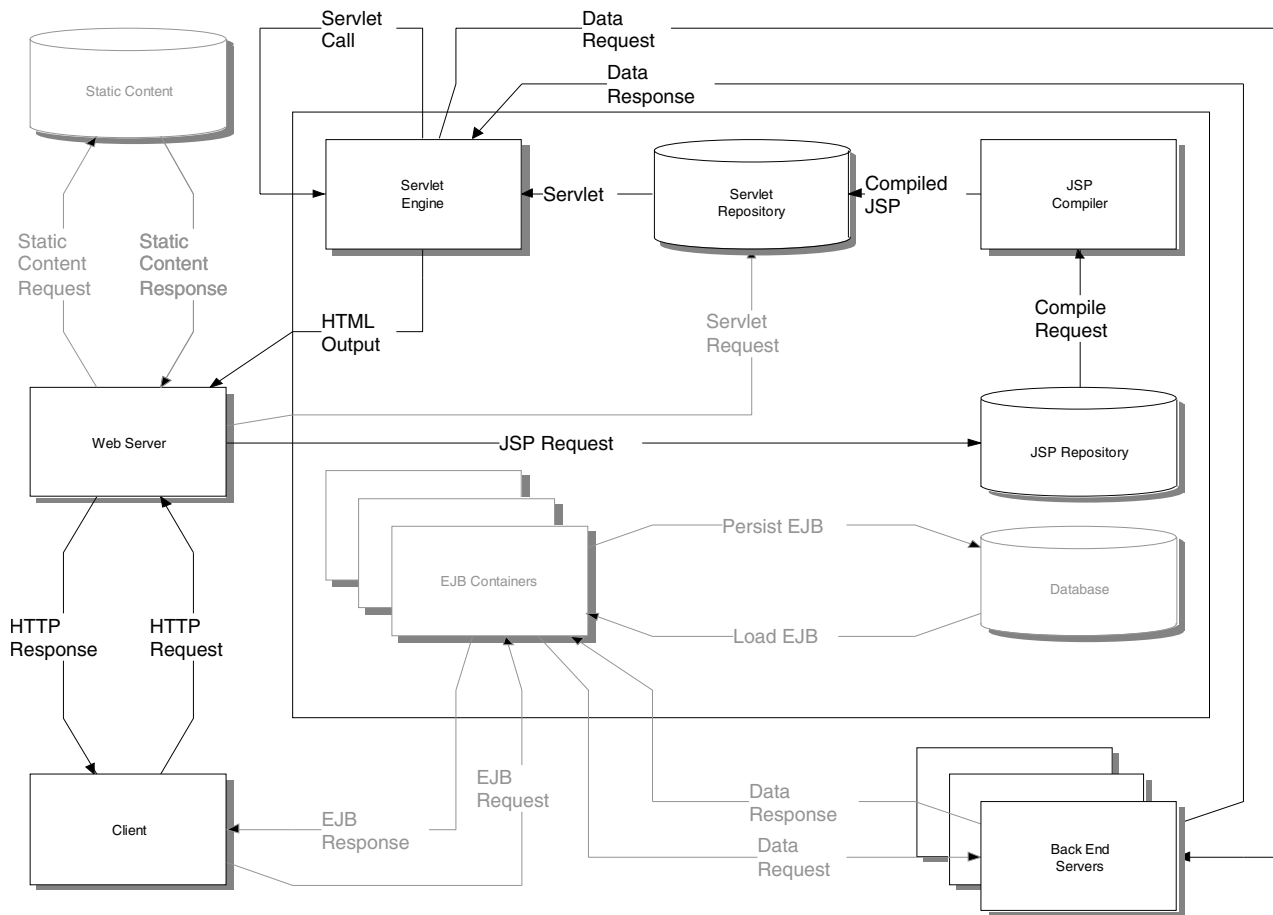


Figure 7. JSP Request Data Flows

If the JSP consists only of HTML tags, the servlet produced by the compiler simply sets the correct fields on the response, opens an output stream to the client and writes the HTML. If other JSP tags are used then the compiler will create Java code in the servlet to perform the requested functions as well as writing the static pieces of the page. The other tags may include snippets of Java

code or directives to access back-end servers. See 3.2, “Java Server Pages” on page 120 for more details on creating and running Java Server Pages.

1.7.4 EJB Interactions

Note: Enterprise Java Services, which is the WebSphere component that provides Enterprise Java Bean Support, is not included in WebSphere Application Server Version 2.x Standard Edition.

Enterprise Java Beans are the most powerful feature of the WebSphere Application Server environment. They build on the philosophy that the application programmer should not have to worry about the infrastructure, but merely about the application. This means that the server that deploys the EJBs must provide a number of services to EJBs in order to manage them properly. These services are provided by an entity called an EJB Container.

Figure 8 shows a simplified diagram of client EJB interactions:

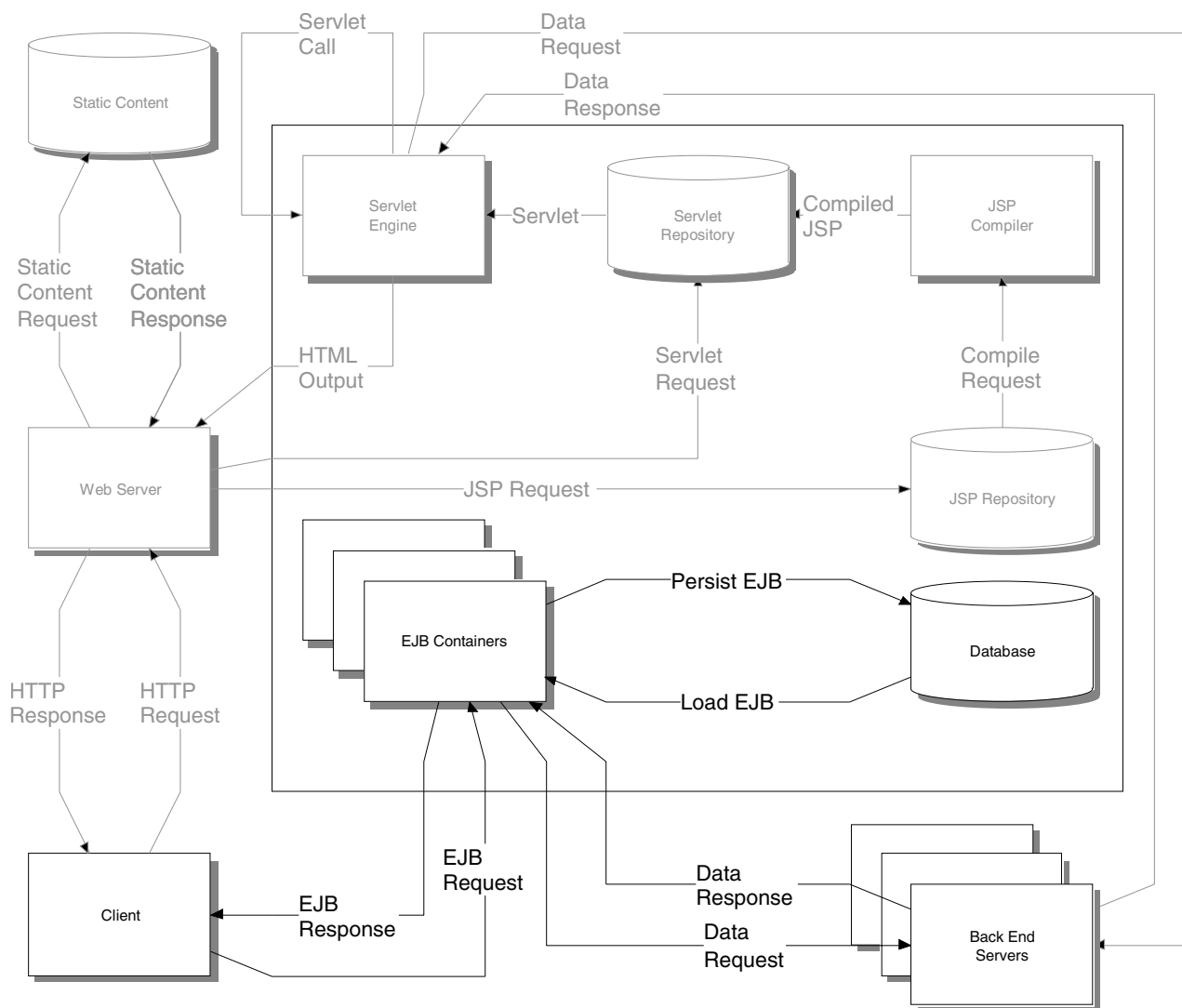


Figure 8. EJB Data Flows

The Enterprise Java Beans specification (see <ftp://ftp.javasoft.com/docs/ejb/ejb.10.pdf>) defines two types of Enterprise Java Beans: Session Beans (see 1.7.4.4, “The Life Cycle of a Stateful Session Bean” on page 25) and Entity Beans (see 1.7.4.3, “The Life Cycle of an Entity Bean” on page 22). WAS supports both types of EJB. Session beans are further divided into *stateless* and *stateful* session beans while entity beans are further divided into beans with *Bean Managed Persistence* (BMP) and *Container Managed Persistence* (CMP).

1.7.4.1 EJB Architecture

Figure 9 shows a diagram of client-to-EJB interaction in more detail:

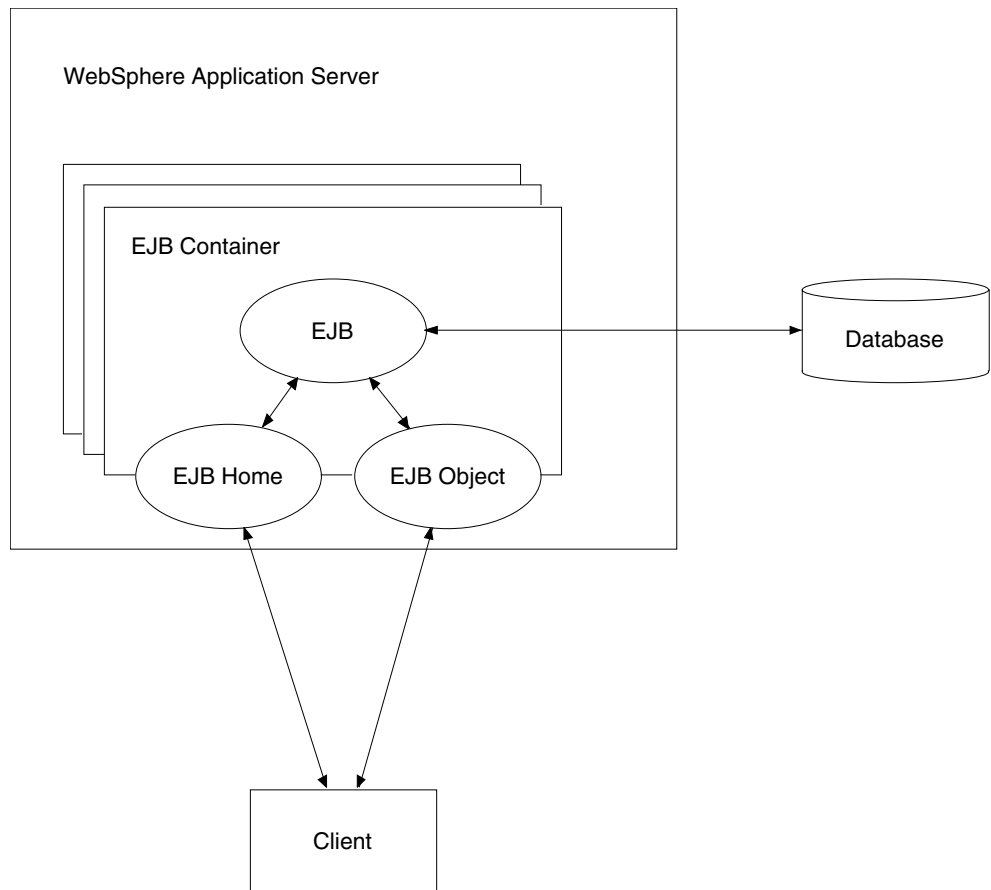


Figure 9. EJB Interaction Detail

In order for the management of EJBs to be handled by the server properly, a client must access EJBs only through a proxy provided by the EJB container. This allows the container to control persistence, security, caching and connection management with no knowledge by the client that all of these functions are occurring with no code in the EJB to control these functions. In order to facilitate this, all client access to EJBs is done by means of instances of the EJB Home and the EJB Object interfaces, that are created by the developer during development. This allows the server to perform management tasks under the covers by mapping the calls to these interfaces to appropriate calls to the EJB

itself and also by calling infrastructure methods on the EJB to control transactions and storage to databases.

The EJB Home interface instance is responsible for allowing clients to find and create EJBs. For entity beans the home interface includes methods for finding single beans or groups of beans based on certain criteria including at least one method that allows the location of a bean in a database using a primary key class. For both entity and session beans the home interface includes methods to create new instances of an EJB inside the container and return a reference to an EJB Object interface instance for the bean. Note that there is one EJB Home interface instance per class of EJB in a container, but there may be many EJB Object interface instances depending upon how many actual instances of the EJB class are present.

The EJB Object interface is responsible for providing access to the operations of an EJB. Each call to an EJB Object interface instance is mapped to a corresponding call to a bean instance by the container, subject to security considerations. Because of the separation from the actual bean, the container is free to release resources used by the bean, such as database connections or even the bean instance itself to other uses and restore the EJB instance when a call is made to it by a client.

1.7.4.2 Steps in Using an EJB

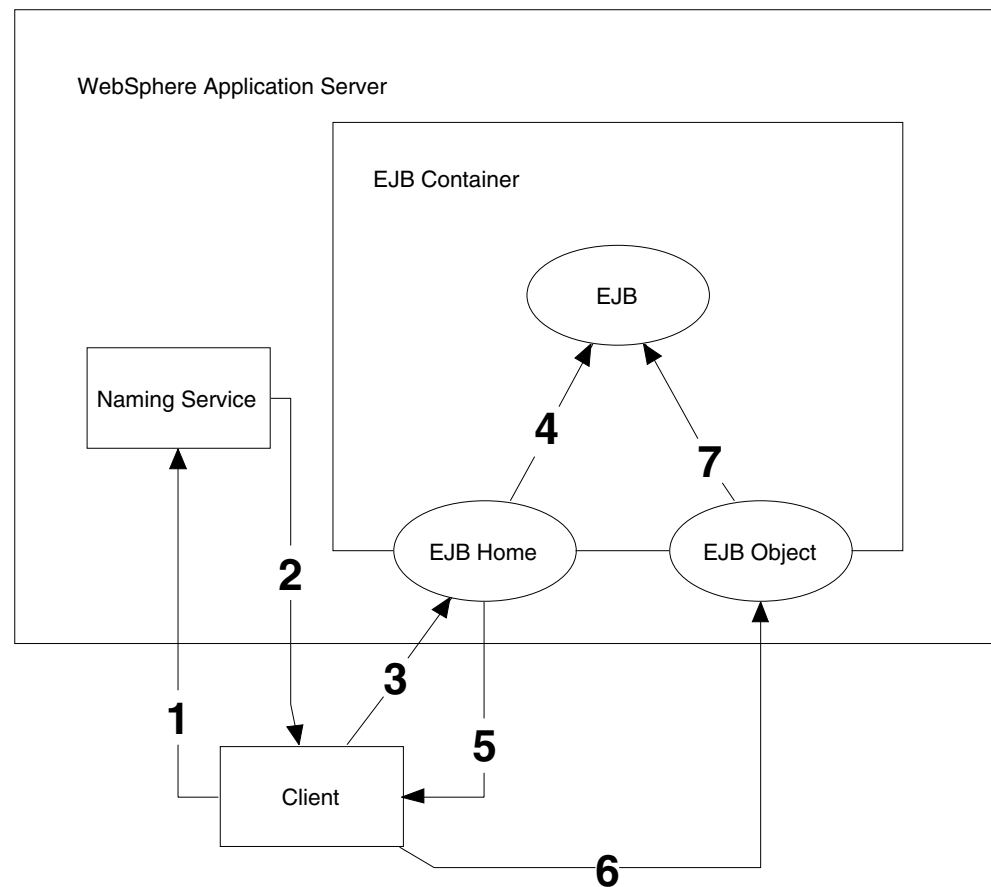


Figure 10. Steps Used to Call Methods on an EJB

Figure 10 shows the steps involved in a client accessing an EJB:

1. The client requests from the naming service (provided as one of the components of WebSphere) a reference to the EJB Home interface of a particular class of EJBs.
2. The naming service replies with the location of the Home interface instance for the EJB class in the container in which the EJB is deployed.
3. The client performs either a create (for a new bean instance) or a find (for an existing entity bean instance) on the EJB Home interface instance.
4. The EJB Home interface instance locates or creates the EJB instance and places it in the container and creates the EJB Object interface instance.
5. The EJB Home interface instance replies to the client with a reference to the EJB Object instance.
6. The client calls methods on the EJB Object interface instance to access business logic on the EJB.
7. The EJB Object interface instance calls the corresponding methods on the EJB while the container manages the resources needed to accomplish this task.

1.7.4.3 The Life Cycle of an Entity Bean

Entity Beans are so called because they typically represent entities, objects with a persistent state. The Enterprise Java Beans specification (see <ftp://ftp.javasoft.com/docs/ejb/ejb.10.pdf>) defines an entity bean as a typical entity object with the following characteristics:

- It represents data in the database
- It is transactional
- It allows shared access from multiple users
- It can be long-lived (lives as long as the data in the database)
- It survives crashes of the EJB server; a crash is transparent to the client

A typical EJB server and container provide a scalable run-time environment for a large number of concurrently active entity objects.

Entity beans typically represent objects with a persistent state, that is data that needs to be remembered for an indefinite period. They are usually stored as a single row in a relational database table. The server controls whether or not the data from the database table is loaded into memory. Similarly the server controls how many entity EJBs are instantiated in the container and which entities they represent.

Entity beans come in two flavors: beans with container managed persistence (CMP) and beans with bean managed persistence (BMP). With CMP beans, the person deploying the bean specifies which fields in the bean are to be stored in a database and the deployment tools generate code to load and store the EJB. With BMP beans the developer adds code to standard methods in the EJB Object interface to load and save the bean to the database. The advantage of CMP over BMP is that it requires less code to be written but the disadvantage is that it provides less flexibility than BMP.

The EJB server manages entity beans in such a way as to keep a logical separation between the instance of the bean running in the container and the entity in the database it represents. This allows the container to maintain a pool of

entity beans available for use and to load the data for different entities in the database into specific bean instances as needed. To put it another way, the server may at any time tell an instantiated entity bean to unload its data to the database and load the data for some other entity. This allows the management of a pool of instantiated entity beans and accompanying resources to maximize the performance of the server transparently to both the EJB developer and the client.

To look at this in more detail look at the state transition diagram shown in Figure 11 on page 24. Each entity bean can be in one of three states:

1. Does Not Exist: The entity bean does not exist in memory at all.
2. Pooled: The entity bean has been instantiated, but it does not represent any particular entity at this time.
3. Ready: The entity bean has been instantiated in memory, and it represents a particular entity.

Note: BMP beans and CMP beans have the same life cycle as shown by the diagram, the only difference being in the way the database code is generated.

To put a bean into the Pooled state from the Does Not Exist state, that is to create an instance ready for the loading of data, the server calls the EJB methods `newInstance()` to create the bean, and `setEntityContext()` to give the bean a reference to the container in which it exists. Conversely, if the server decides that there are too many beans in the Pooled state it calls `unsetEntityContext()` and `finalize()`. The server has complete control over the number of beans that exist in the Pooled state and creates them or destroys them at will based on performance criteria. In WebSphere the performance criteria is set by the server developers and is not configurable by server administrators.

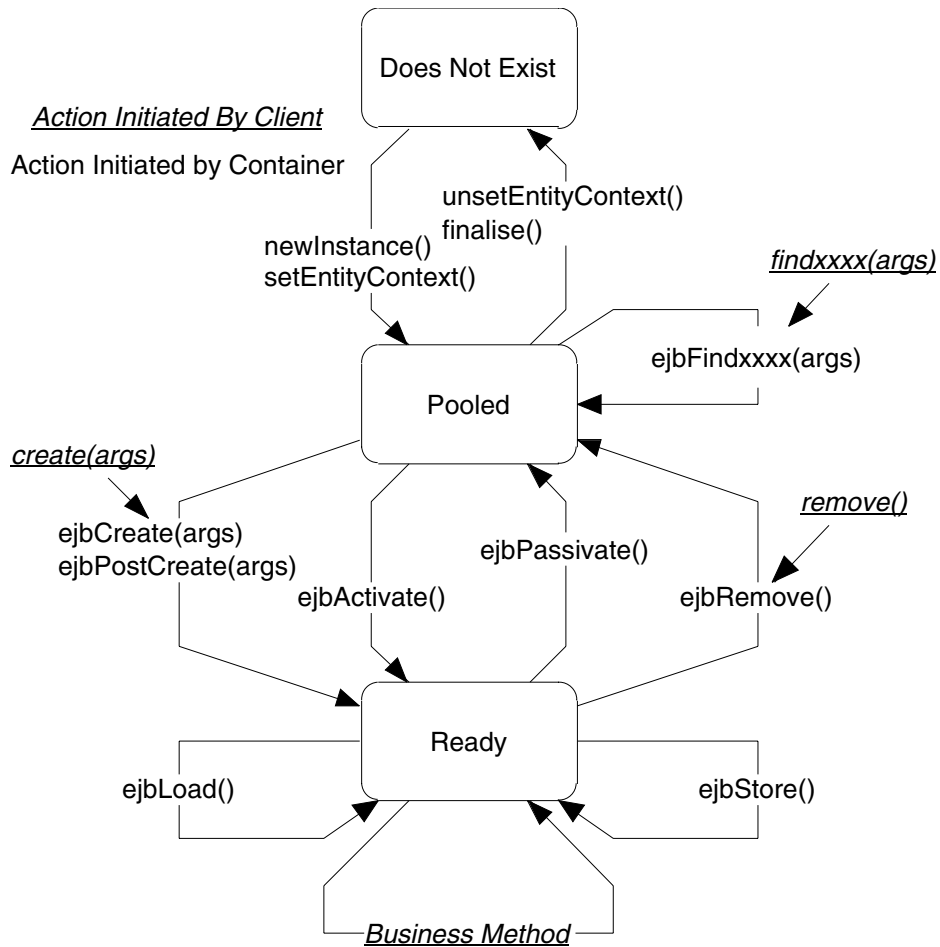


Figure 11. Entity Bean Life Cycle State Transition Diagram

Beans that are in the Pooled state can have several things happen to them. As mentioned previously they can be selected for removal and destroyed by calling `unsetEntityContext()` and `finalize()`. If the client calls one of the find methods to locate a bean or group of beans, then one of the beans in the Pooled state is selected by the server as all of the find methods are static, that is, they do not refer to a particular instance of the bean class but rather to the class as a whole. If the client calls a create method then a bean is selected from the pool to hold the data for the new entity being created and the appropriate `ejbCreate()` method is called with the arguments passed by the client followed by the `ejbPostCreate()` method. It is now put in the Ready state ready for use by a client. Finally, if the server decides that a bean needs to be made ready, potentially as the result of a client calling one of the business methods on the EJB Object interface instance, the `ejbActivate` method is called on the pooled bean, which causes it to load all of its data from the database and become ready.

Beans that are in the Ready state are associated with an entity and ready for use. If a client calls a business method on the EJB Object instance the bean business method is called and the bean remains in the Ready state. Similarly, the `ejbLoad()` and `ejbStore()` methods may be called by the server to load and store information respectively in the underlying database. This is used by the server to synchronize the information with the database as needed. The only way that the

bean can be returned to the Pooled state is if the server decides, based on certain criteria, to *passivate* the bean by calling its `ejbPassivate()` method. This has the effect of causing the bean to write all of its data to the database and return to the Pooled state.

All of this interaction is transparent to the client and to the EJB programmer (provided all of the required methods are implemented correctly) with the server making the decisions. This allows the server to optimize the performance of the system by allocating resources to particular EJBs as required.

WebSphere provides transaction support for entity beans in accordance with the EJB specification (see <ftp://ftp.javasoft.com/docs/ejb/ejb.10.pdf> Section 11, page 95). These transaction characteristics can be specified through the use of the deployment descriptor in the EJB JAR file. See 4.2.4, “Working with Deployment Descriptors Using the Jet Tool” on page 197 for details on how to specify transaction attributes for entity beans and methods.

1.7.4.4 The Life Cycle of a Stateful Session Bean

The first of the two types of session beans is the so-called *stateful* session bean (as opposed to stateless session beans (see 1.7.4.5, “The Life Cycle of a Stateless Session Bean” on page 27). Stateful session beans are designed for client interactions where there may be a number of interactions with the bean by a single client and there needs to be data or *state* stored between interactions. This is often referred to as a session with the EJB.

There are two important differences to note between stateful session beans and entity beans, which could also be said to store state information. The first is that the state does not persist and is not stored anywhere outside of a particular session. This means that if the session is ended for any reason such as when WebSphere is restarted or the session exceeds a timeout value (see “Working with Session Beans” on page 200 for details on how to set this), then the session information will be lost. The second difference is that session beans are designed to be used by a single client whereas entity beans are designed to be used by multiple clients. Thus, there are no “findXXXX” methods on a session bean’s EJB home interface for clients to find particular session beans, only create and remove methods.

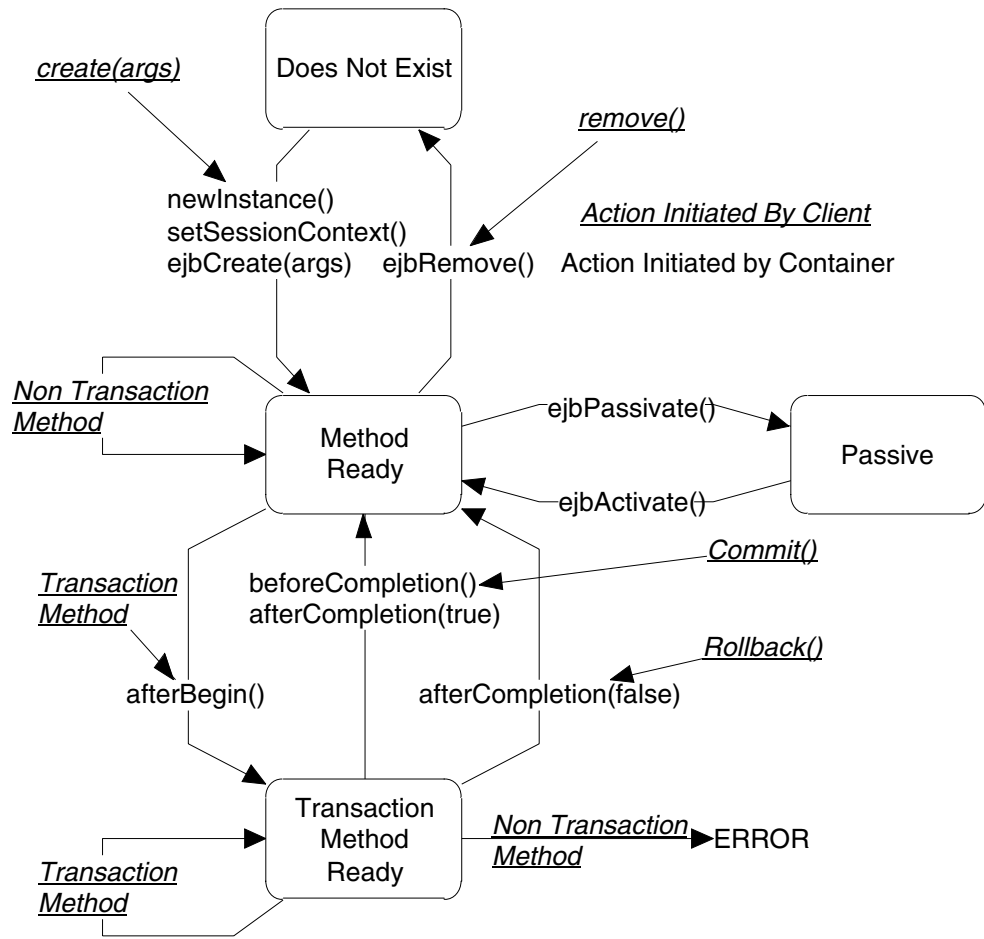


Figure 12. Stateful Session Bean State Transition Diagram

The other important note about stateful session beans is that their transactional state (for example, whether the bean is currently involved in a transaction) restricts the possible method calls that are allowable. Each method has a transaction attribute that is defined either at the bean level or at the method level (see “Working with Session Beans” on page 200 for details on how to specify transaction attributes). If a method must execute in the context of a transaction it is referred to as a transactional method. If a method cannot execute in the context of a transaction then it is a non-transaction method. There are methods that can execute either in the context of a transaction or without one.

Figure 12 is a state transition diagram that shows the possible states that a stateful session bean may be in on the server. The possible states are:

- The bean does not exist.
- The bean is method ready: No transaction has been created on the bean and it is ready to accept method calls.
- The bean is transaction method ready: A transaction has been created for the bean and it is ready to accept transaction method calls.
- The bean is passive: The server has decided to release resources used by the bean and move it to a Passive state.

From the Does Not Exist state the server calls the `newInstance()` method to create a new stateful session bean, the `setSessionContext()` method, to attach the bean to its container and the `ejbCreate(args)` method to instantiate the bean in the container in response to a `create(args)` method call made by the client. This puts the bean in the Method Ready state.

From the Method Ready state a number of things can happen. If the client calls `remove()`, then the server calls `ejbRemove()` and the bean is removed. If the server decides that it needs to free resources then it might passivate the bean by calling `ejbPassivate()` in which case the bean is moved to the Passive state. Business methods that do not require transactions may be called, in which case the bean is returned to the Method Ready state. Finally, a transaction method may be called that causes the container to create a new transaction and then to call the `afterBegin()` method. The business method is then called and the bean is moved to the transaction method Ready state.

While the bean is in the transaction method Ready state, only methods that can execute on transactions may be called. If a non-transaction method is called then the server issues a `java.rmi.RemoteException`. This includes things like the client calling `remove()`, which is considered a non-transaction method, or a non-transaction business method. It is also an error to call a method that requires a new transaction that is different from the one in which the bean is currently executing. When a transactional business method is called the bean returns to the transaction method Ready state. If the client issues a `commit()` then the server calls the `beforeCompletion()` method on the bean to allow it to write any information to the database and then `afterCompletion(true)`. If the client calls `rollback`, then `beforeCompletion()` is not called and `afterCompletion(false)` is executed. Note that the bean will not be passivated while in the transaction method Ready state.

If the bean is in the Passive state then the server may activate it in response to a request from the client to execute a business method.

Stateful session beans may have a timeout value specified that limits the amount of time a session can be kept open by a client before the server can reclaim the bean instance for other uses. Only beans in the Method Ready and Passive states can be timed out. Beans with current transactions, for example, those in the transaction method Ready state, will not be timed out on the server.

1.7.4.5 The Life Cycle of a Stateless Session Bean

Life for a stateless session bean is much simpler than that of the other sorts of beans. The server maintains a pool of stateless session beans and creates new ones by calling the `newInstance()`, `setSessionContext()` and `ejbCreate()` methods. If a request to execute a business method comes in, the server selects a session bean instance from the method ready pool to service the client request and executes the method. Finally the server removes beans from the method ready pool as required by calling the `ejbRemove` method. Figure 13 on page 28 shows a diagram of this interaction.

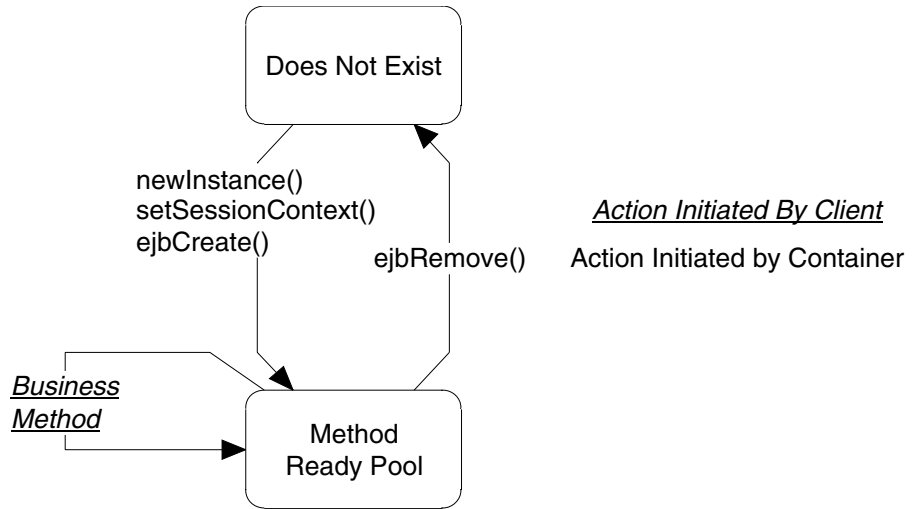


Figure 13. Stateless Session Bean State Transition Diagram

Chapter 2. Installation of WebSphere and Associated Products

WebSphere Application Server provides an environment for running applications on the Web. These applications often require services from other products such as Web and database servers. This chapter describes the steps needed to install WebSphere Application Server and to configure it to work with the associated products. We don't cover the installation processes for other products, but only some configuration setup that is related to the WebSphere environment. In this document, we limit our discussions to the Windows NT and AIX platforms. We used the Standard and Advanced Editions of the WebSphere V2.02.

Before you install WebSphere, you have to make sure that your system has met the hardware and operating systems requirements that are described in 1.4, "Planning for WebSphere Standard Edition" on page 9 and 1.5, "Planning for WebSphere Advanced Edition" on page 10.

A typical application environment consists of:

- A Java engine
- A Web server
- The WebSphere Application Server
- A back-end database
- A development environment
- A Site Analyzer
- A front-end client

Since WebSphere requires a Java engine and updates your Web server environment, you should install WebSphere only after your Java engine and Web server have been installed and set up. Other components (for example, Site Analyzer or VA Java) can be installed independently.

We don't explain configuring the Web server in great detail. We discuss only the basic things that are necessary to build a Web site.

After all of your components are installed, you should configure the related parameters for each component in order to make them work together. The following sections describe that process.

2.1 Infrastructure Installation for Windows NT V4.0

The products that we installed were:

- JDK 1.1.6
- HTTP Server V1.3.3
- Domino Go Webserver
- Netscape SuiteSpot V3.x
- Microsoft IIS V4.0
- DB2 UDB
- Domino R5 Server

2.1.1 JDK 1.1.6

The Java Development Kit (JDK) contains a basic set of software and libraries that are needed to compile, debug and run Java applets and applications. A separate package without the tools is available in the Java Runtime Environment (JRE). Developers might want to use the JRE for testing applications in a similar environment that end users would run.

The original JDK implementation is available from Sun. You can download it for free according to the license agreement, from Sun's Java site at <http://java.sun.com>. As of now, JDK 1.2, which is also known as JDK 2.0, is available. Since Java is undergoing a rapid change, the JDK is incorporating more and more APIs. Your JDK might not include the latest APIs. In that case, you have to obtain them separately. WebSphere V2.02 does not support JDK 2.0.

IBM has its own JDK implementations on AIX, OS/2, OS/390, OS/400, VM/ESA and Win32 platforms. The reason for having these vendor-specific implementations is for optimization, stability, and IBM service and support on these platforms. The JDKs are available from the IBM JCentral site at <http://www.ibm.com/java/jdk/>. Since IBM delivers JRE only on the Win32 and OS/2 platforms, end users for other platforms should already have the required level of JDK installed in their production system.



Figure 14. The IBM JCentral JDK Site at <http://www.ibm.com/java/jdk/>

When running applications on WebSphere platforms, we recommend that you use the IBM JDK implementations. It will ensure a robust and smooth integration with WebSphere products.

2.1.1.1 Installation and Setup Procedure

On Win32 platforms, the JDK is distributed in a zip file or in an InstallShield self-extracting file. The zip format file does not provide download languages other than English.

To install the JDK, you have to download the appropriate file to your hard disk. After that, simply execute the InstallShield file or unpack the zip file into the appropriate directory.

The installation program adds an entry to the Windows registry that specifies *JavaHome* directory (see Figure 15 on page 31). WebSphere will read these registry entries to locate JDKs in your system. You can install more than one JDK version. In this case, the registry will contain several JDK version entries. The "CurrentVersion" key indicates and sets the active JDK version.

```

\HKEY_LOCAL_MACHINE\SOFTWARE\JavaSoft\Java Development Kit\
    Name: "CurrentVersion" Data: "1.1"
\HKEY_LOCAL_MACHINE\SOFTWARE\JavaSoft\Java Development Kit\1.1\
    Name: "JavaHome" Data: "D:\jdk1.1.6"
  
```

Figure 15. Registry Entries for JDK Installation

Since the installation program does not create or alter the CLASSPATH environment variable, you have to add the CLASSPATH variable using **Start > Settings > Control Panel > System > Environment** and put in the appropriate CLASSPATH values (see Figure 16 and "System CLASSPATH Environment Variable" on page 32).

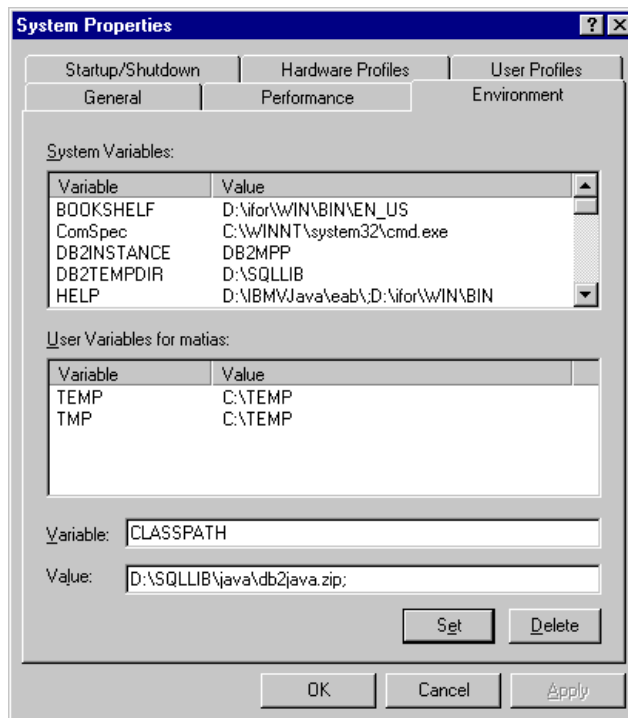


Figure 16. Setting the CLASSPATH Variable

Note: If you install WebSphere V2.02 from a CD, JDK V1.1.6 is located on the CD and it can be invoked from the WAS setup program.

2.1.1.2 Additional Java Libraries

Developers might require additional Java APIs for developing applications using some of the advanced features of WebSphere Application Server. The DB2/UDB

package delivers JDBC drivers and SQLJ libraries. The WebSphere Application Server incorporates JSDK, JSP, JNDI, XSL, XML4J and WebSphere specific class libraries. Table 5 on page 32 lists APIs with the JDK, DB2/UDB and WebSphere package. You can also obtain information and download these APIs and the latest APIs from the JavaSoft site at <http://www.javasoft.com> or from the Sun Java Developer Connection site at <http://developer.java.sun.com>.

Table 5. Java API Package Names and Locations

API	Package Name	Location
JDBC	java.sql	[JDKRoot]\lib\classes.zip
DB2 JDBC driver	com.ibm.jdbc	%DB2PATH%\java\db2java.zip
DB2 SQLJ support	com.ibm.sqlj	%DB2PATH%\java\sqlj.zip %DB2PATH%\java\runtime.zip
JSDK	javax.servlet	[WASRoot]\lib\jsdk.jar
JSP	com.sun.server.http	[WASRoot]\lib\jst.jar
JNDI	javax.naming	[WASRoot]\lib\jndi.jar
Lotus XSL	com.lotus.xsl	[WASRoot]\lib\lotusxsl.jar
XML4J	com.ibm.xml org.xml	[WASRoot]\lib\xml4j.jar
IBM WebAS servlet	com.ibm.servlet	[WASRoot]\lib\ibmwebas.jar
IBM Data Access Beans	com.ibm.db	[WASRoot]\lib\databeans.jar
IBM EJS	com.ibm.CORBA com.ibm.ejb com.ibm.ejs com.transarc.encina javax.ejb javax.jts org.omg.CORBA	[WASRoot]\lib\ejs.jar [WASRoot]\lib\ejscientruntime.jar

2.1.1.3 System CLASSPATH Environment Variable

The JDK will automatically locate its default Java library, classes.zip. You do not need to specify it in the CLASSPATH variable. WebSphere will let you choose between its own CLASSPATH or the system CLASSPATH. It will automatically recognize paths to its Java libraries. WebSphere will not work if you include the paths to the system CLASSPATH.

Set or verify your system CLASSPATH to include the remaining libraries:

- %DB2PATH%/java/db2java.zip;%DB2PATH%/java/sqlj.zip;
%DB2PATH%/java/runtime.zip or other database JDBC/SQLJ driver libraries
- Paths to additional JAVA API class libraries either in directory, JAR or zip formats
- Path to your project class libraries for development purposes

2.1.2 HTTP Server V1.3.3

For the Windows NT platform, the IBM HTTP Server is distributed on the WebSphere CD or you can download it from the IBM Web site,

<http://www.software.ibm.com/webserver>. The installation program, which is `setup.exe`, will automatically install the Web server into your system. During installation, you will have to supply a user ID to run the service as shown in Figure 17. You can always change it later by using the **Start > Setting > Control Panel > Services > Startup** dialog box (see Figure 18 on page 34).

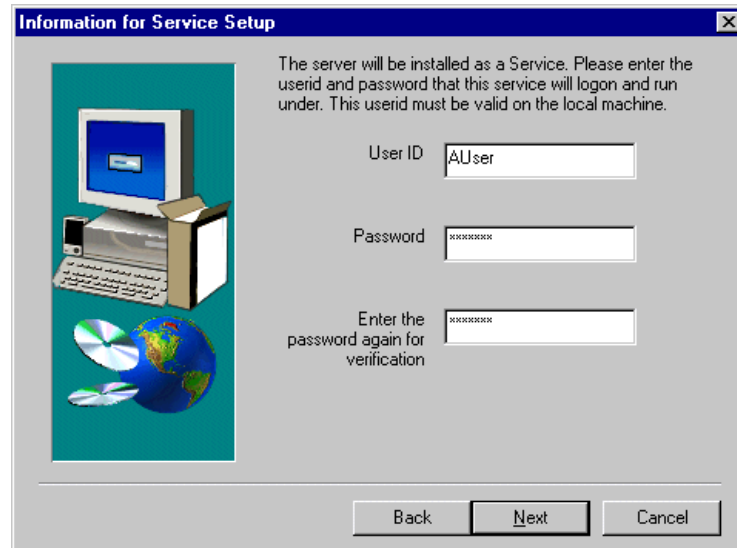


Figure 17. Specifying the User ID to Run IBM HTTP Server

The installation adds some entries into the registry as shown below:

```
\HKEY_LOCAL_MACHINE\SOFTWARE\IBM\HTTPServer\1.3.3
Name: "FolderName" Data: "IBM HTTP Server"
Name: "PATH" Data: "C:\Program Files\IBM\GSK\lib"
Name: "ServerRoot" Data: "C:\Program Files\IBM HTTP Server"
```

The Web server is installed as a service. By default it is set to run automatically after you reboot your system. You can change this behavior by setting the Startup Type to Manual in the **Start > Setting > Control Panel > Services > Startup** dialog box as shown in Figure 18. You can start or stop the Web server by either:

- Invoking the Windows start menu program using:
Start > Program > IBM HTTP Server > Start / Stop
- Or entering the following command in the Windows console:

```
net start "IBM HTTP Server"
```

or

```
net stop "IBM HTTP Server"
```



Figure 18. Specifying IBM HTTP Server to Manually Start Up Using a Specific Account

2.1.3 Server Modules

The IBM HTTP Server follows the Apache Web server architecture by using modular structures. The HTTP Server consists of modules, each of which provides a particular service. This approach simplifies the addition or modification of services in the server. Each module is implemented by a single object file or a DLL file. The LoadModule directives in the configuration file specifies which module needs to be loaded as an active module. When the Web server starts, it scans the configuration file and loads every module specified by the LoadModule directive.

The default server configuration consists of several base modules. Other than these base modules, many third-party vendors provide various modules for different type of services. WebSphere provides its service on the HTTP Server by using a plug-in module. It is implemented using a module named mod_ibm_app_server which corresponds to the DLL file mod_ibm_app_server.dll.

2.1.3.1 Configuring IBM HTTP Server

You can set the Web Server configuration in the <ServerRoot>\conf\httpd.conf file. This file controls how the server runs. Server parameters are specified as *directives*. A directive can be either a one-line directive or a section directive. Section directives can also be nested. For example:

```
# A Comment ..
# One line directive

DirectiveName directive-values

# Section directive

<SectionName SectionParams>
Directive1 directive1-value
Directive2 directive2-value
...
</SectionName>
```

The installation process provides you with the default directives values that are sufficient to make the server run. You can also specify names, addresses, additional modules, access structure and security. To do that, there are some basic directives that have to be set up. You can obtain these other directives either from the IBM HTTP Server online configuration manual or from the Apache site at <http://www.apache.org>.

ServerAdmin Directive

This directive specifies an e-mail address where clients can write for information. This e-mail address normally appears in any error message that is sent to the client. The syntax of this directive is:

```
ServerAdmin webmaster_email_address
```

ServerName Directive

This directive sets your site's host name according to this directive.

```
ServerName your.host.name
```

ServerRoot Directive

The server will find all resources and configuration files relative to a root directory.

```
ServerRoot directory_absolute_pathname
```

DocumentRoot Directive

This directive gives an absolute path to the root of your document tree. Any URL that refers to URL paths other than *Alias* will be mapped relative to the document root.

```
DocumentRoot document_root_absolute_pathname
```

Alias Directive

This directive allows you to map any URL path to the given directory path name.

```
Alias url_path directory_absolute_pathname
```

LoadModule Directive

The LoadModule directive lists a module as a server active module to be loaded at Web server initialization. It also specifies actual module implementation using <filename>. In the Win32 platform, a file name is normally a DLL file name.

```
LoadModule module_name filename
```

Deny Directive

The server will deny all access from specified hosts.

```
deny from all | host_list
```

Allow Directive

The server will grant access from specified hosts.

```
allow from all | host_list
```

Order Directive

This directive specifies the order of access checking:

- Verify Deny lists first: [deny,allow].
- Verify Allow lists: [allow,deny].
- Grant access to any host that appears in the Allow list but not in the Deny list [mutual-failure].

```
order [deny,allow] | [allow,deny] | [mutual-failure]
```

Directory Directive

This section directive groups a set of directives that apply to the specified directory.

```
<Directory directory_name> .. </Directory>
```

Limit Directive

This section directive groups a set of directives that apply to specified access methods. The method can be GET, POST, PUT, DELETE, CONNECT or OPTIONS.

```
<Limit access_method_lists> .. </Limit>
```

Locate Directive

This section directive groups a set of directives that apply to a given URL.

```
<Location URL> .. </Location>
```

2.1.3.2 Configuring httpd.conf for WebSphere Applications

The default WebSphere installation modifies the httpd.conf file. It adds a LoadModule directive to register the plug-in module `ibm_app_server_module` as an active module. The installation also specifies a default alias as well as module parameters:

```
LoadModule ibm_app_server_module
D:/WebSphere/AppServer/plugins/nt/mod_ibm_app_server.dll
Alias /IBMWebAS/samples/ "D:/WebSphere/AppServer/samples/"
Alias /IBMWebAS/ "D:/WebSphere/AppServer/web/"
NcfAppServerConfig BootFile
D:\WEBSPH~1\APPSER~1\properties\bootstrap.properties
NcfAppServerConfig LogFile D:\WebSphere\AppServer\logs\apache.log
NcfAppServerConfig LogLevel TRACE|INFORM|ERROR
```

You can change or add additional directives as needed. You can set the host name, change the document tree structure, create new aliases, set access control or update user authentication.

2.1.3.3 Controlling User Access

Apache provides two methods for controlling access to documents:

1. Domain level access control
2. User Authentication

You can use these methods to provide server-wide or per-directory access control:

- Server-wide access control is set using a global Access Control File (ACF) in <ServerRoot>/conf/access.conf.
- Per-directory access control is set using <Directory>, <Limit> and <Location> section directives, or per-directory ACFs.

Domain Level Access Control Example

To limit access to documents under *your.domain.name/secret/docs* URL to only GET or POST requests and to deny all access except from a specific domain, use a Location section directive:

```
<Location /secret/docs>
<Limit GET POST>
order deny,allow
deny from all
allow from some.domain.com
</Limit>
</Location>
```

User Authentication Example

To limit access to documents to only GET requests and deny all access except for the users tony, tims and johny, in domain *some.domain.com* use a Directory section directive:

```
<Directory /www/html/secret/docs>
AuthUserFile /system/passwords/passwordfile
AuthType Basic
<Limit Get>
require user tony tims johny
order deny,allow
deny from all
allow from some.domain.com
</Limit>
</Directory>
```

2.1.4 Domino Go Webserver

Domino Go Webserver is a Web server product from Lotus. It is also known as the IBM Internet Connection Server. The product is available on many platforms, including Windows NT, AIX, Solaris, HP/UX, OS/2 and OS/390. Lotus has integrated the Webserver into its Lotus Domino R5 server family. It is the Web services component in the Domino R5 Application Server product. For Windows NT platforms, the Web server was included in the WebSphere 1.0 package. For later versions of Websphere, you need to obtain the Domino Go Webserver installation package separately.

2.1.4.1 Configuring Domino Go Webserver

Domino Go Webserver provides Web-based configuration and administration forms. You can access these online forms by accessing

<http://<your.domain.name>/admin-bin/cfgin/initial>, as shown in Figure 19.

These forms are easy to use with a complete description and examples for each

parameter. As an alternative, you may configure the Web server directly using its configuration file: httpd.conf.

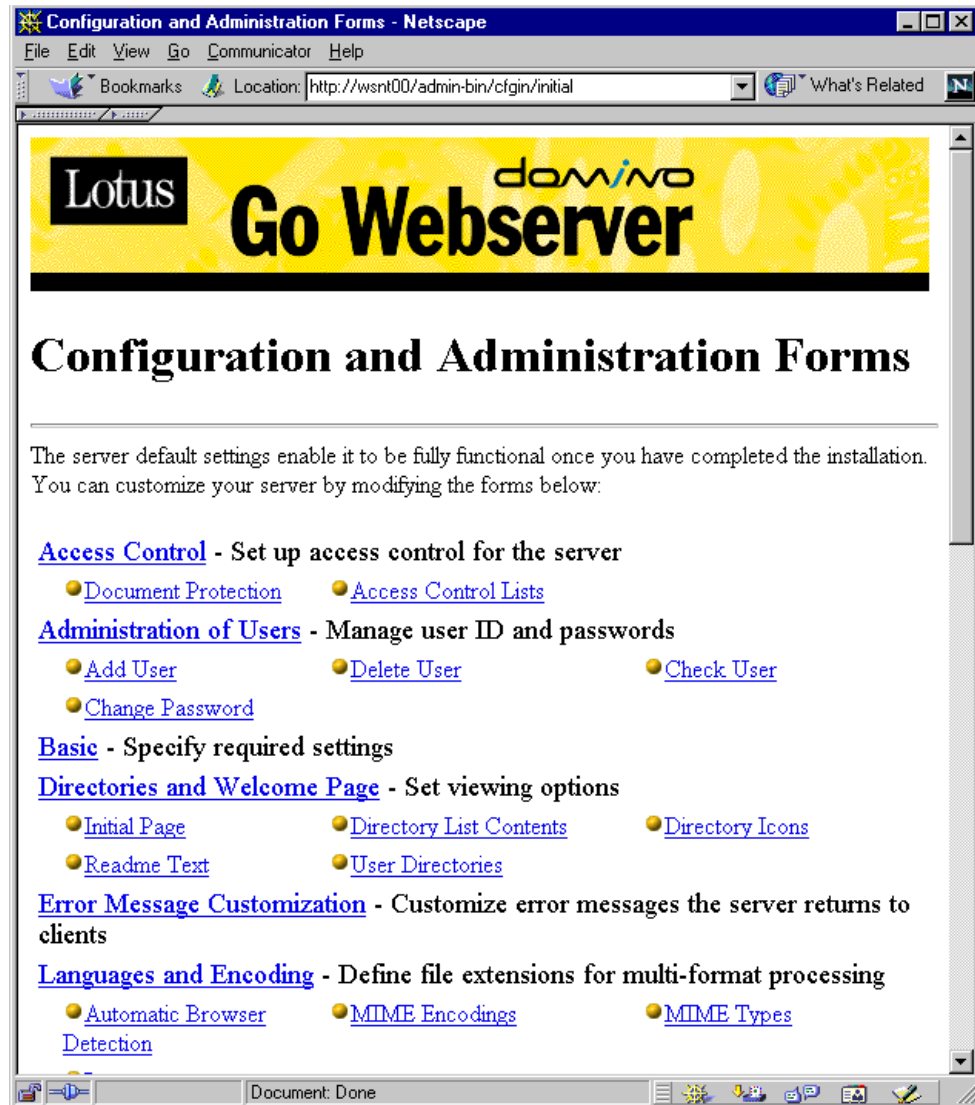


Figure 19. Domino Go Webserver Configuration and Administration Forms

The configuration file, httpd.conf, specifies Web server parameters by using directives. Table 6 lists several important directives. They need to be tailored according to your Web site's requirements.

Table 6. Lotus Domino Go Webserver Basic Directives

Directive	Syntax	Description
ServerRoot	ServerRoot <path>	Set Web server root directory to <path>.
Hostname	Hostname <your.domain.name>	Set Web server host name.
Port	Port <port>	Set port used by the Web server.
Exec	Exec <url_template> <path>	For executable resources (such as cgi programs), map <url_template> to <path>

Directive	Syntax	Description
Map	Map <url_template> <another_url>	Forward any request for <url_template> to <another_url>
Fail	Fail <url_template>	Block access to <url_template>
Redirect	Redirect <url_template> <fully_qualified_server_url>	Forward any request for <url_template> to fully qualified server URL
Pass	Pass <url_template> <path>	Translate any request for <url_template> as an access to <path> .
Protection	Protection <prot_name> { <prot_directives> }	Define a protection rule setup, <prot_name> , by using <prot_directives> rules. The rules can be constructed using UserID, GroupID, ServerID, AuthType, GetMask, PutMask, PostMask, DeleteMask, Mask, ACLOverride, PasswdFile, or GroupFile directives.
Protect	Protect <url_template> <prot_name>	Protect <url_template> with protection rules <prot_name>.

Note: <url_template> and <another_url> are relative to <your.domain.name> URL. They can specify a particular Web document or use wild cards such as *, *.gif, or *.html. The <path> is a fully qualified physical directory name such as C:\WWW\admin, C:\WWW\html.

Some important settings that need to be configured in constructing a Web site:

- Basic settings: host name, default port number, and server root. To configure these parameters select **Basic** in the Configuration and Administration forms. These settings correspond to Hostname, Port and ServerRoot directives.
- Mapping rules that map a request URL to a physical file or to another URL. Use the *Directories and Welcome Pages* administration forms. These settings corresponds to Exec, Map, Fail, Redirect and Pass directives.
- Access control to protect documents and authenticate users. Use the *Access Control* administration forms. These settings correspond to Protection and Protect directives in the httpd.cnf file.
- Manage users using Administration of Users in the Configuration and Administration page.

As an example, look at the following sections of the httpd.cnf file:

```
# This is a comment

# The Domino Go Webserver root directory
ServerRoot C:\WWW\Bin

#Web server fully qualified hostname
Hostname wsnt00.itso.ral.ibm.com

#Web server port
```

Port 80

```
...
# Define protection rules setup.
Protection MYPROJECT-PROT {
    PasswdFile C:\WINNT\admin.pwd
    Mask All@(*)
    PostMask All@(*)
    PutMask All@(*)
    GetMask All@(*)
    AuthType Basic
    ServerID Private_Authorization
}

# Protect these resources using MYPROJECT-PROT setup.
Protect /myproject/* MYPROJECT-PROT
Protect /mydocs/*.pdf MYPROJECT-PROT

...

#
# Mapping rules: Exec, Map, Redirect. They are applied in the order they appear
# in this file.

Exec /admin-bin/* C:\WWW\Admin\*
Exec /cgi-bin/* C:\WWW\CGI-Bin\*

# Map WebSphere samples using Map then Pass directives.
Map /sample/* /IBMWebAS/samples/*
Pass /IBMWebAS/samples/* D:\WebSphere\AppServer\samples\*

# Redirect URL for WebSphere Administration Tool.
Redirect /WASAdmin http://wsnt00.itso.ral.ibm.com:9527

# Don't touch my project source code.
Fail /myproject/*.java

# This is for my project.
Pass /myproject/* D:\WebSphere\AppServer\servlets\XtremeTravel\*

# Pass the rest to look for in c:\WWW\html directory.
Pass /* C:\WWW\html

...
```

2.1.4.2 WebSphere Plug-In for Domino Go Webserver on Windows NT

WebSphere can run on top of Domino Go Webserver by using a plug-in. For Domino Go Webserver 4.x or later on Windows NT, the plug-in file is go46.dll.

During installation the WebSphere installation program adds several entries into the httpd.cnf file:

```
ServerInit D:\WebSphere\AppServer\plugins\nt\go46.dll:init_exit
D:\WebSphere\AppServer\properties\bootstrap.properties
ServerTerm D:\WebSphere\AppServer\plugins\nt\go46.dll:term_exit
Pass /IBMWebAS/samples/* D:\WebSphere\AppServer\samples\*
Pass /IBMWebAS/* D:\WebSphere\AppServer\web\*
```

The first and second entries map server initialization and termination functions into a corresponding API entry in the go46.dll. The other entries are Websphere default mapping rules.

2.1.5 Netscape SuiteSpot V3.x for Windows NT

Netscape SuiteSpot is a family of server products used to create a complete Web site. SuiteSpot server products include the Enterprise Server, Fasttrack Server, Messaging Server, Directory Server, Proxy Server, and Media Server. Netscape Enterprise is the Web server product in the SuiteSpot family. Since we worked with installing a Web server for WebSphere, we limited our discussion to the Enterprise Server. You can obtain more information about Netscape SuiteSpot products at <http://www.netscape.com/servers>.

When you install your first Netscape SuiteSpot server product, the Installation process creates the Administration Server. You have to supply an administrator port number, and a superuser's user ID and password. In the next SuiteSpot server installation, you won't have to re-create another Administration Server. You can use the superuser for granting administrator privilege to another user. For details on the Netscape SuiteSpot installation, you can refer to the SuiteSpot installation documentation.

The Administration Server is a tool that is used for configuring any Netscape SuiteSpot server product. In the Administration Server, each SuiteSpot server has its own Server Manager, which has the same look and feel for every server. A Server Manager has a collection of forms that you use to configure the SuiteSpot server.

The Administration Server is a Web-based service using a predefined port number. To access this facility, you should be the Administration Server superuser or have an administrator user ID. The superuser privilege allows you to access all Server Manager forms. The administrator ID limits access only to the Server Manager forms for a specified SuiteSpot server. Using a Web browser, you can enter the URL of your domain followed by administrator port number. For example, if your domain name is `www.your-domain.com` and the port number is 8301, you can browse to:

`http://www.your-domain.com:8301`

2.1.5.1 Setting Up a Netscape Enterprise Web Server

To set up a Web server, you need to be an Administrator superuser. After successfully authenticating the user ID, the server will bring up the Server Administration page as shown in Figure 20.



Figure 20. Server Administration Page in the Netscape Administration Server

In the General Administration section of this page, you can see a list of servers, grouped by product name, along with the server's status and its name. In this section, you can either create, remove or configure servers. If you click **Create New Netscape Enterprise Server 3.61**, the server will show a Server Manager form for Netscape Enterprise Server Installation, as shown in Figure 21.

Figure 21. Netscape Enterprise Server Setup Form

In this form, you can set up a Web server by providing the following information:

- The Server Name specifies your fully qualified domain name.
- The Bind address specifies an IP address that is not the default IP address to be used for this Web server. This applies only if the machine supports multiple IP addresses.
- The Server Port specifies the TCP port number where the Web server should listen. The port number must be unique for each SuiteSpot server.
- The Server Identifier specifies the name of the server as seen in the Server Administration page.
- The Document Root specifies the root directory for your Web documents.

After completing the form, click **OK** to create a new Web server.

2.1.5.2 WebSphere Plug-In for Netscape Enterprise on Windows NT

The WebSphere Application Server can run on top of a Netscape Enterprise Web server by using a plug-in module. You should install this plug-in only after you have set up the corresponding Netscape Enterprise server. In the WebSphere installation process, the installation program will ask you to select the appropriate Web server plug-in (see also Figure 42 on page 66). It will also ask for the location of the Enterprise configuration file (see Figure 44 on page 67).

The Netscape Enterprise configuration file is:

```
<SuiteSpotRoot>\https-<ServerIdentifier>\config\obj.conf
```

The WebSphere installation program will automatically insert the following lines inside the obj.conf file:

```
Init fn="load-modules" funcs="init_exit,service_exit,term_exit"  
shlib="D:/WebSphere/AppServer/plugins/nt/ns35.dll"  
  
Init fn="init_exit"  
bootstrap.properties="D:/WebSphere/AppServer/properties/bootstrap.properties"  
  
NameTrans from="/IBMWebAS/samples" fn="pfx2dir"  
dir="D:/WebSphere/AppServer/samples"  
  
NameTrans from="/IBMWebAS" fn="pfx2dir" dir="D:/WebSphere/AppServer/web"
```

The first line indicates that in the Web server initialization, the Web server should load the WebSphere plug-in module, ns35.dll. The second line tells the Web server to invoke init_exit method for initializing the module using specified bootstrap properties. The third and fourth line specify directory aliases for the WebSphere default document directories.

Normally, the obj.conf file is set by the Administration Server only. Therefore, after installing WebSphere, you should apply these changes into the Web server. You can do this by clicking the **Apply** button on any Server Manager form for the corresponding Web server.

2.1.5.3 Setting Up Directory Alias

The Directory Alias function maps a URL path name, relative to the domain name, into the path name of the directory that contains Web documents. To set up the Directory Alias in Netscape Enterprise, you should log in to the Administration Server as either a superuser or a Web server administrator. On the top of the Server Manager form, click the **Content Management** button to invoke the Content Management form, as shown in Figure 22. For now, we are concerned with three of the items on the left side of the window:

1. Primary Document Directory - establishes the root directory for your Web document primary tree. This root directory maps to your domain URL.
2. Additional Document Directories - specifies aliases into the root of secondary document trees.
3. URL Forwarding - maps a URL path name, relative to the domain name, into another URL.

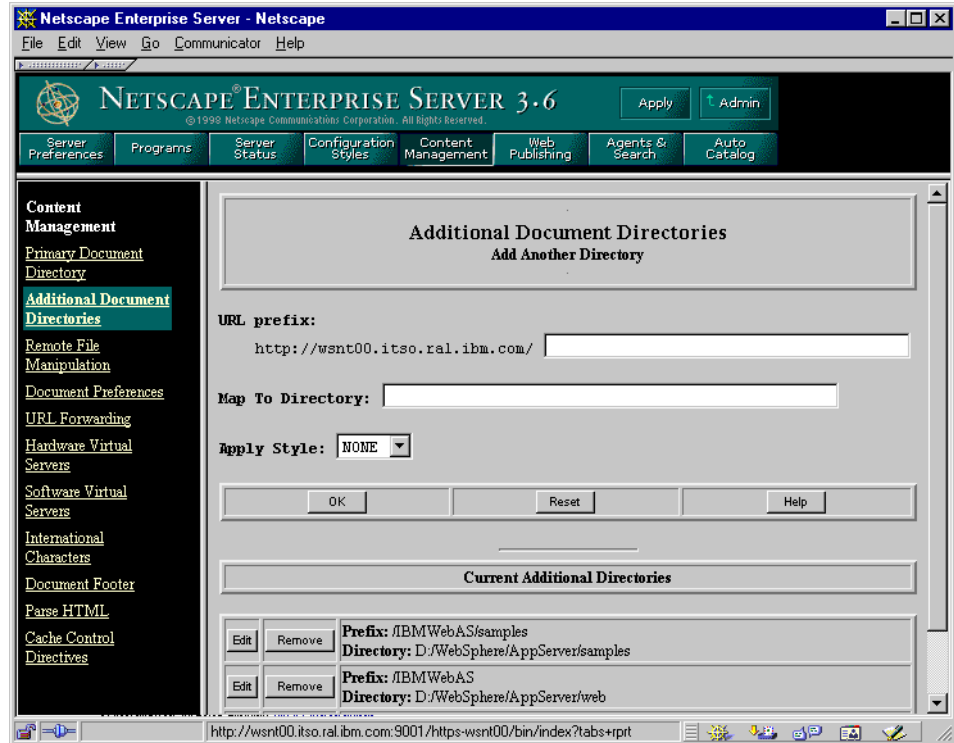


Figure 22. Content Management Form for Setting Up Directory Alias

2.1.5.4 Virtual Servers

Virtual servers are servers that have different addresses but refer to the same Web server. You can set up virtual servers in the Content Management form shown in Figure 22. Netscape provides you with two ways of creating this feature:

1. Hardware Virtual Servers

Several servers with different IP addresses (possibly with different domain names) are located in the same machine and served by the same Web server. To provide Hardware Virtual Servers, your machine must support multiple IP addresses. Netscape Enterprise maps the Virtual Server IP address into the Virtual Server root document directory.

2. Software Virtual Servers

Several servers with different domain names are served by the same machine, the same IP address and served by the same Web server. Netscape Enterprise maps Virtual Server domain name, in URL format, into the Virtual Server root document directory.

2.1.5.5 Setting Up Access Control

Access Control determines which users can access specific Web documents and what actions they can perform. To configure access control in Netscape Enterprise, you log on to the Administration Server as a superuser or as the Web server administrator. In the top frame of the Server Manager form, click the **Server Preferences** button and select **Restrict Access** in the left-hand frame. This will bring up the Access Control List Management form, as shown in Figure 23 on page 46. The Access Control List (ACL) is a set of rules that controls user access to Web documents. Netscape Enterprise lets you create resource-based ACLs or named ACLs.

- To create a resource-based ACL, choose the resources that you want to protect in the *Pick a resource* field, by browsing the directory or supplying file type wildcards. After that, click the **Edit Access Control** button to invoke the Edit Access Control form as shown in Figure 24 on page 47.
- To modify an ACL, select the ACL from a drop-down list in the *Pick an existing ACL* field, and click the corresponding **Edit Access Control** button.
- To create a named ACL, enter an ACL name in the *Type in the ACL name* field and click the corresponding **Edit Access Control** button.

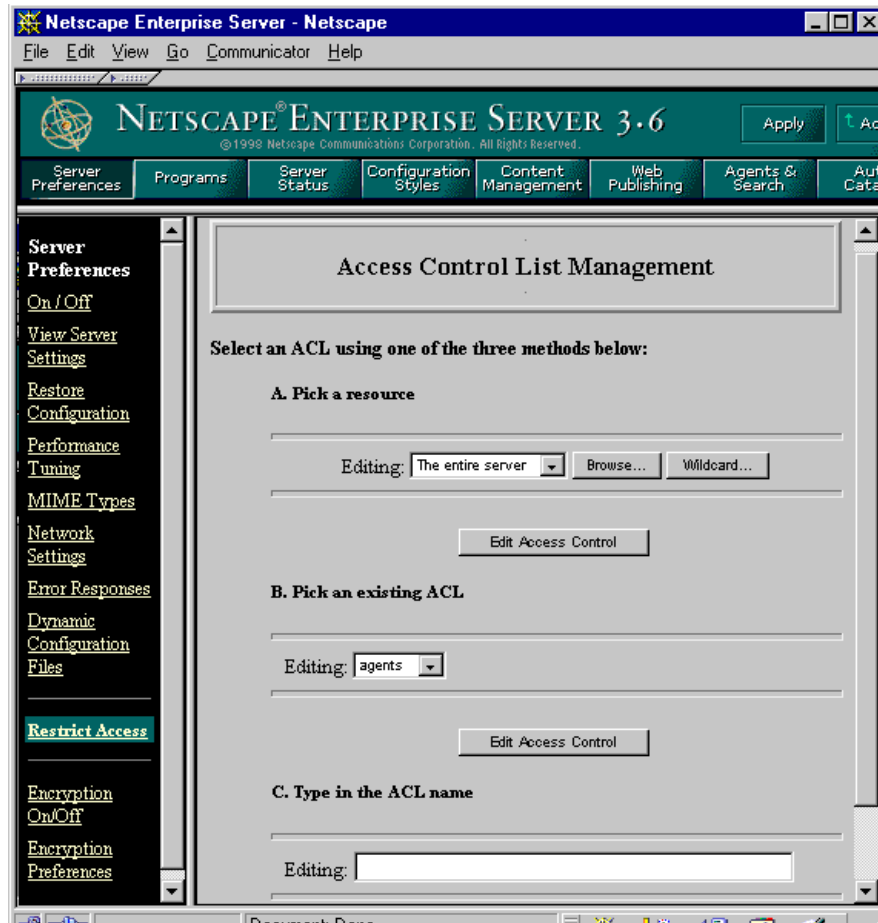


Figure 23. Restrict Access Form for Controlling User Access

When you click one of the Edit Access Control buttons, the server will show the Edit Access Control form (Figure 24 on page 47). This form allows you to specify Access Control rules. The form consists of two frames. The top frame is for specifying rules, the bottom frame is for selecting parameter values.

The Access Control rules are specified by:

- Specifying rule action
 - Allow or deny action for users or machines that match the rule's conditions.
- Specifying rule conditions that are based on:
 - User/Group

You can specify that the Web server authenticates the user with the user ID and password against an authentication database. You can limit access to only specified users or groups.

- Machine

You can enter host names or IP addresses as rule conditions.

- Specifying access rights for the following operations: read, write, execute, and delete.

After specifying the rule, click **Submit** to activate the rule.

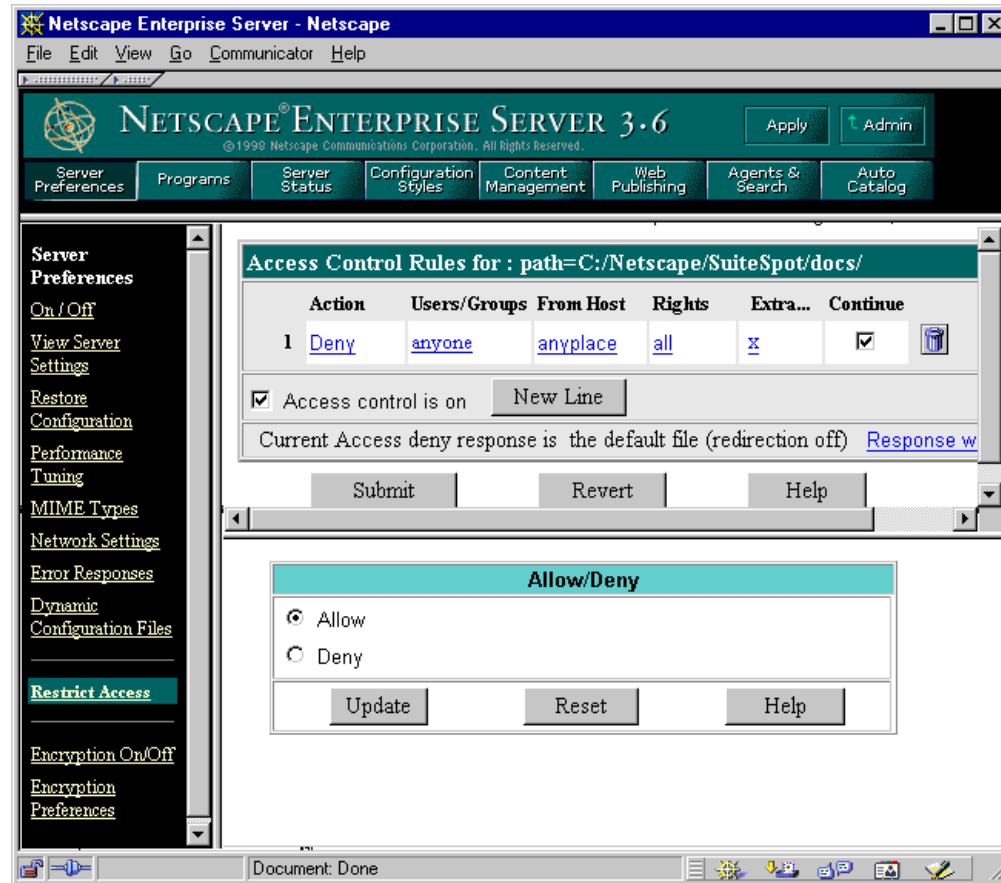


Figure 24. Editing Access Control Rules

2.1.6 Microsoft IIS V4.0

Microsoft Internet Information Server (IIS) is the Web server product in the Microsoft Windows family. IIS V4.0 can be installed with the Windows NT 4.0 Option Pack. In this document, we don't discuss the installation process for IIS. We describe only some items that may be used in setting up the WebSphere environment. You can read the IIS manuals that are available as HTML documents from the Windows NT task bar.

Note: WAS supports IIS 3.0 but the Internet Service Managers interface is different.

2.1.6.1 Managing Microsoft IIS

IIS provides easy-to-use utility tools for managing Internet services. The tool comes in two forms:

1. As a part of Microsoft Management Console (MMC) (see Figure 25 on page 48), which you can access from **Start > Program > Windows NT 4.0 Option Pack > Microsoft Internet Information Server > Internet Service Manager**.

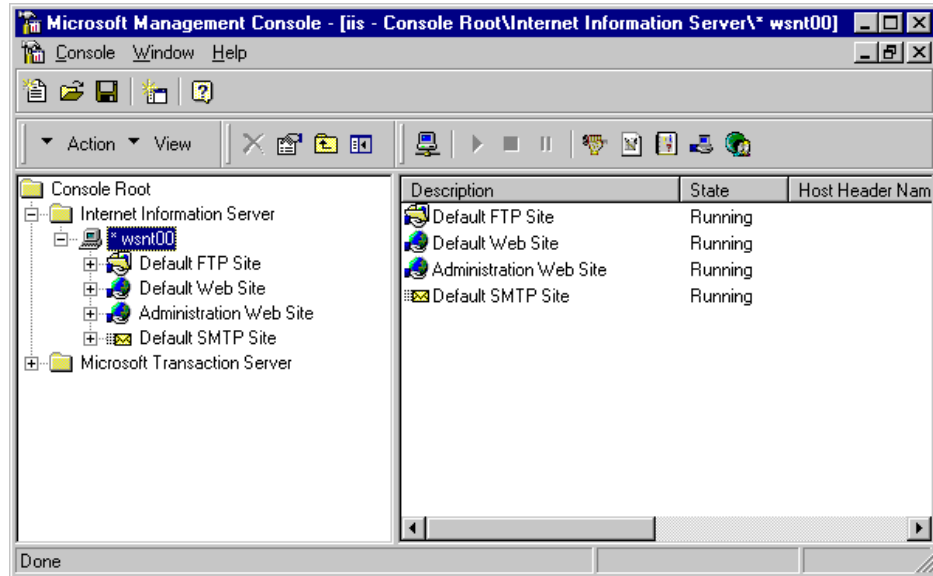


Figure 25. Internet Service Manager Functions in Microsoft Management Console

2. As an IIS Internet Service manager Web application (Figure 26 on page 49), which you can access by browsing the IIS Administrator site <http://localhost:<port>>. The <port> is the IIS Administrator port number. If you do not know this port number, you can see it in the right frame of the Microsoft Management Console.

The Web version of Internet Service Manager provides the same functions as the Microsoft Management Console.

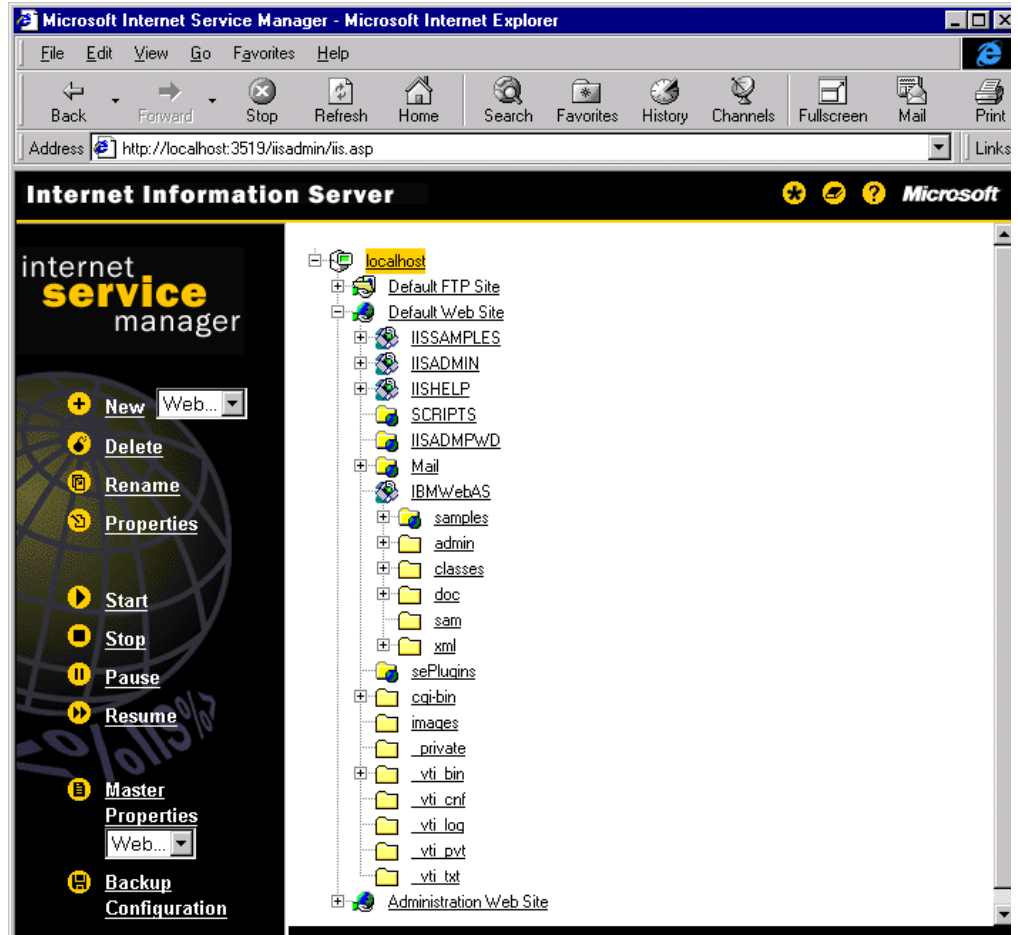


Figure 26. IIS Internet Service Manager

2.1.6.2 Setting Up a Web Server

To set up a Web server, you can use either IIS Microsoft Management Console or the Internet Service Manager Web application (Figure 26 on page 49). If you are using the Web version of Internet Service Manager, the right frame of the Internet Service Manager shows site structures for all sites in a particular machine (localhost). To create a new Web server, you should be the local administrator and perform the following steps:

- Select **localhost** in the right frame, then click **New Web** in the left frame.
- Enter the name of the new Web site and click **OK**.
- You should see your new Web site on the right frame of the Internet Service Manager.
- At this point, you still can't start the Web site, since you have not bound it to a Web address.
- Double-click the new Web site to bring up the Web Site Identification form (Figure 27 on page 50). Enter the IP address and port number for this Web site. Then click **Save**.

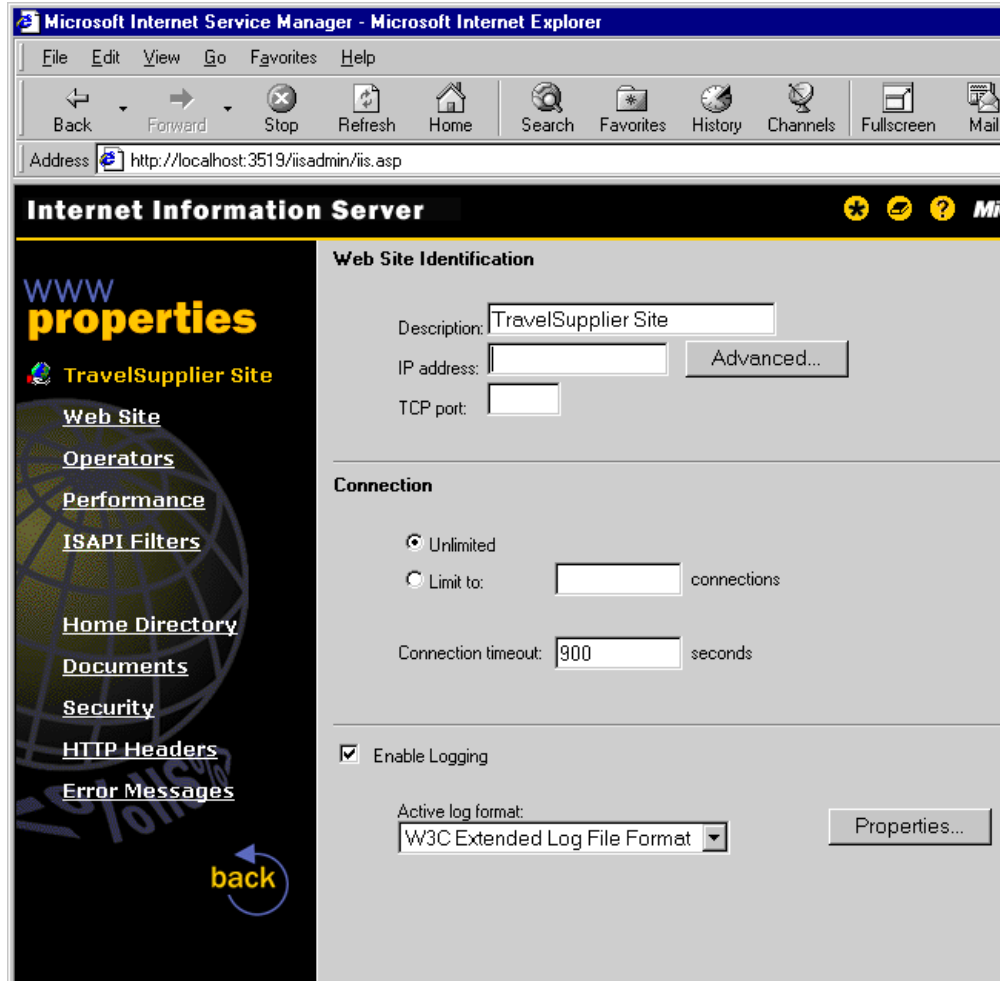


Figure 27. Binding Web Address to New Web Site in IIS Internet Service Manager

- Your new Web site is now ready to be used. However, you still need to add Web documents to this new site. This is described in the following section.

2.1.6.3 Configuring a Web Server

To configure a Web server, you can use either the IIS Microsoft Management Console or the Internet Service Manager Web application. If you are using the Web version of Internet Service Manager to configure the Web site directory, perform the following steps:

- Double-click the Web site entry in site structures, then select **Home Directory** on the left frame to bring up the Directory Properties form (Figure 28 on page 51).
- This form defines the mapping of a request to a particular physical location to another URL. You can:
 - Map the request to a physical directory in the local machine.
 - Map the request to a shared directory in another machine.
 - Redirect it to another URL.

- In this form, the term *Application* represents the Document Root. The *Starting Point* defines the root URL for this particular Document Root. The name in the square brackets represents the root URL of the Web site.

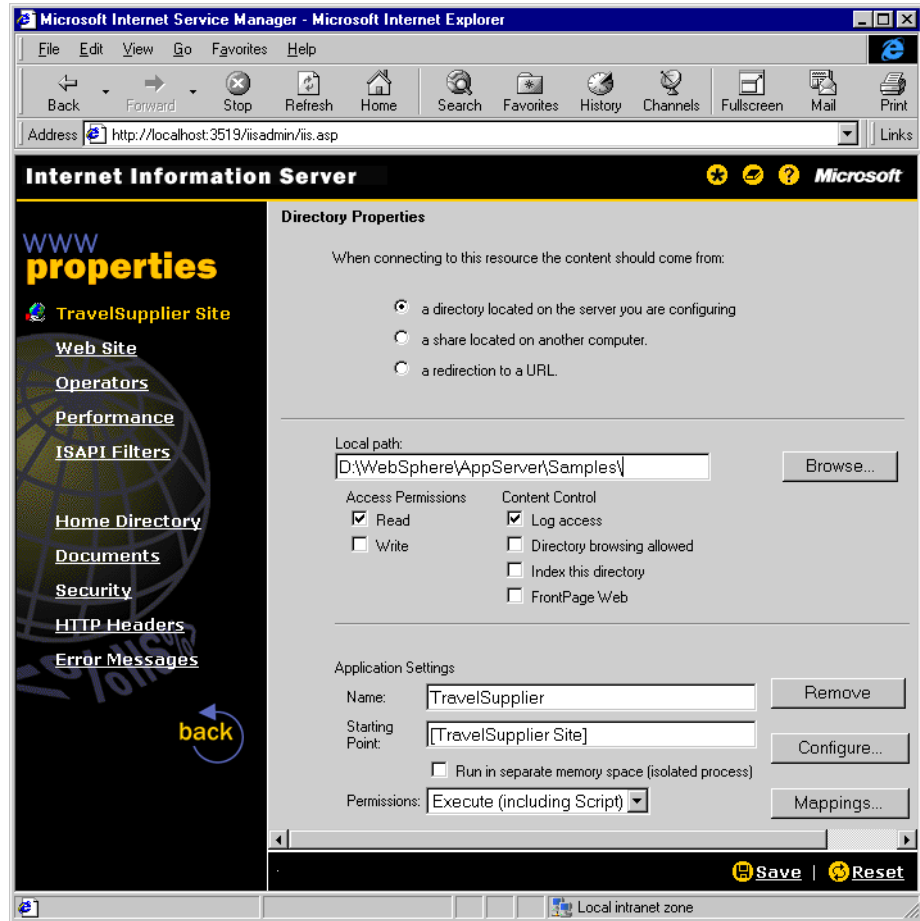


Figure 28. Setting Directory Properties in IIS Internet Service Manager

2.1.6.4 WebSphere Plug-In for Microsoft IIS

WebSphere runs on top of IIS by using a plug-in. The plug-in for IIS is iis20.dll, which is loaded when IIS starts. The WebSphere installation program will add the following entries into the Windows registry:

```

\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Parameters\VirtualRoots
    Name: "/IBMWebAS" Data: "D:\WebSphere\AppServer\web"
    Name: "/IBMWebAS/samples" Data: "D:\WebSphere\AppServer\web\samples,,1"
    Name: "/sePlugins" Data: "D:\WebSphere\AppServer\nt,,5"

\HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere Application Server\2.0
    Name: "iisextensionloc" Data: "/sePlugins/iis20.dll"
  
```

These registry entries identify the location of the plug-in and specify the WebSphere default root document directory.

2.1.7 DB2 Universal Database (UDB) for Windows NT

To work with the WebSphere application environment, the database system must support Java. DB2 UDB comes with full Java support. It provides JDBC and SQLJ drivers for a smooth integration of the WebSphere application server. Section 6.2, “Using DB2 UDB for WebSphere Applications” on page 349 describes techniques for accessing DB2 databases.

If your environment does not have DB2, the WebSphere installation will automatically install DB2 Universal Database (UDB), but it is then limited to only development purposes. For an end-user production environment, you should obtain the DB2 package separately. For more information about DB2 products, you can go to <http://www.ibm.com/data/db2>.

In this document, we don't describe the installation of the DB2 products. We limit our discussion to setting up DB2 in the WebSphere environment and configuring the remote interface between the DB2 Client and DB2 Server components. For more information about DB2 installation and setup guidelines, you can refer to the Quick Beginnings for DB2 manual. This manual is also available as online documentation after the DB2 UDB installation.

2.1.7.1 DB2 Products on Windows NT

DB2 delivers its database components in several products. For the Windows NT platform, DB2 provides DB2 UDB, DB2 Client Application Enabler (CAE) and DB2 Software Development Kit (SDK). Each product contains a set of components. Some of the components are included in more than one product.

DB2 UDB

On Windows NT, DB2 delivers DB2 UDB as the database server product. It contains:

- The DB2 Server component, which provides the database engine and run-time environment
- The DB2 Client component
- The DB2 Communication support component
- DB2 Java support components

In the DB2 environment, applications require DB2 Client components to access the Server component. Practically, the DB2 Client component is the only interface used for accessing the Server component. The applications are considered part of the database client, even if they are running on the application server.

If you install DB2 UDB in the same machine as the Web server, you do not need to install the client software. The DB2 UDB already includes the DB2 Client component in the product.

If DB2 UDB and the Web Server are in different machines, you need a DB2 Client component in the Web server and a remote connection between the client and the DB2 Server. To service remote clients, DB2 uses the DB2 Communication Support component. This communication infrastructure is available in several protocols, including TCP/IP, APPC, NetBIOS and IPX/SPX.

DB2 UDB comes in several editions: DB2 for Workgroup, DB2 Enterprise Edition, and DB2 Extended Enterprise Edition. Currently, the latest edition for Windows

NT is DB2 UD2 5.2. The minimum DB2 level that has Java support is DB2 5.0. The DB2 UDB installation CD also includes DB2 CAE and DB2 SDK.

DB2 Client Application Enabler (CAE)

Database client machines require DB2 CAE to provide DB2 client services. The DB2 CAE contains:

- DB2 Client component
- DB2 Communication support
- DB2 Java support

To support various client platforms, DB2 provides CAE on various platforms. The CAE is platform dependent, but provides the same functions across different implementation platforms. DB2 CAE is available on the DB2 UDB installation CD or from separate media for a particular platform.

DB2 Software Development Kit (SDK)

The DB2 SDK provides software and tools to build database applications using DB2. The DB2 SDK contains:

- DB2 Client component
- DB2 Java support

Since DB2 SDK also includes the DB2 Client component, developers do not need to install CAE separately.

2.1.7.2 Setting Up DB2 in WebSphere Environment

In the WebSphere environment, if you are installing the products separately you should install the DB2 Client and DB2 Java support components. If you installed WAS Advanced from the product CD, DB2 V5.2 is packaged with it and automatically installed. Otherwise, DB2 Client and DB2 Java support are available in the DB2 UDB, CAE and SDK products. If you install from the DB2 UDB installation CD, carry out the following steps:

- Log on as a local administrator.
- Run the DB2 installation program `setup.exe` from the installation CD.
- The installation program will ask you to select the appropriate DB2 products, as shown in Figure 29 on page 54. If the data server and WebSphere are in the same machine, select **DB2 Universal Database**. Otherwise, select either **DB2 Software Developer's Kit** or **DB2 Client Application Enabler**.



Figure 29. Choosing Appropriate DB2 Products during DB2 Installation

Setting Up Java Environments

To use DB2 in the WebSphere environment, you need to add DB2 Java class paths and native implementation library paths into the system environment and WebSphere Java engine setup. The Java path information in the system environment is required for the DB2 Client component or for developing applications.

The class libraries are in the %DB2PATH%\java directory. The native implementation library is in the %DB2PATH%\bin directory. If our DB2PATH is D:\SQLLIB, we can add these paths into system variables using the following steps:

- Invoke Windows NT System Properties dialog by using **Start > Settings > Control Panel > System > Environment**.
- Select **CLASSPATH** from the System Variables list and append the following entries into the Value text field:
`D:\SQLLIB\java\db2.zip;D:\SQLLIB\java\sqlj.zip;D:\SQLLIB\java\runtime.zip`
- Select **PATH** from the System Variable list and append the following entries into the Value text field:
`D:\SQLLIB\bin`
- Click **OK** to save and apply the settings.

To add the path information into the WebSphere Java engine setup, follow these steps:

- Log on to the WebSphere Administration Tool as an administrator.
- On the left frame, select **Setup > Java Engine**. This will bring up the Java Engine setup dialog box as shown in Figure 30 on page 55.
- On the Paths tab, append the following entries into the Application Server Classpath field:
`D:\SQLLIB\java\db2.zip;D:\SQLLIB\java\sqlj.zip;D:\SQLLIB\java\runtime.zip`
- Append the following entry into the User Libpath field:

D:\SQLLIB\bin

- Click **Save** and log out from the Administration Tool.
- Stop and restart the Web server to activate the new settings.

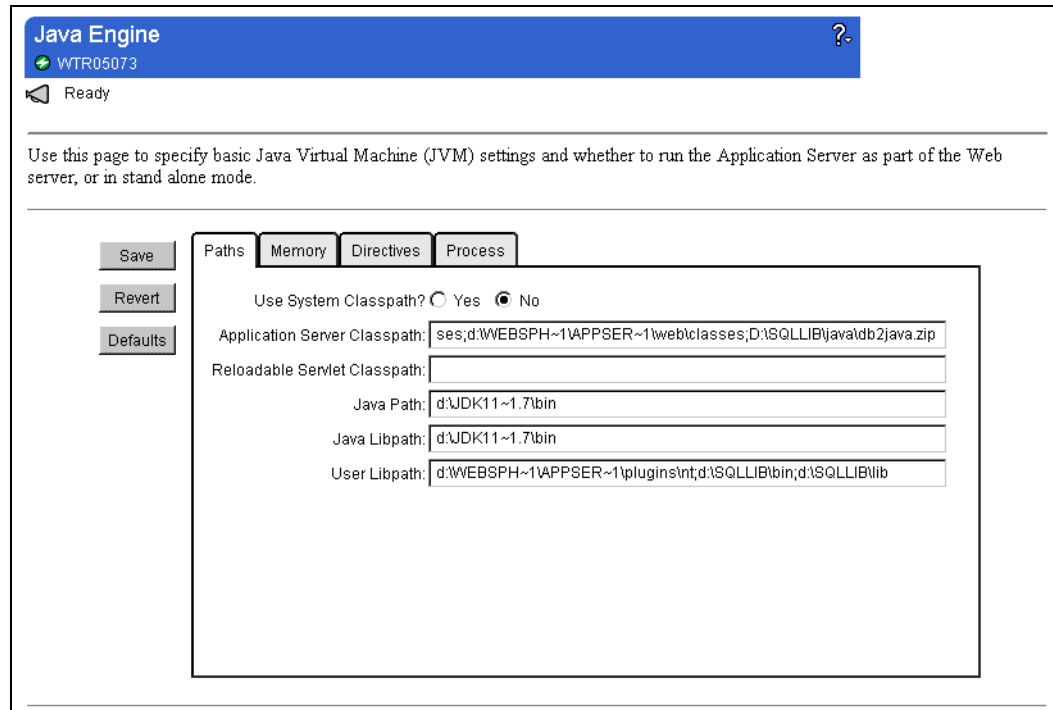


Figure 30. Setting Up DB2 Java Path in WebSphere Administration Tool

2.1.7.3 Configuring Remote Interface

A remote interface is required if the application server and the database server are not in the same machine. The remote interface links the DB2 Client component to a remote database. Configuring a remote interface involves setting up server communication profiles and defining a remote database connection in the client machine.

Configure Server Communication Profile

For the TCP/IP protocol, a DB2 remote interface requires two TCP services: one for the connection service and another for interrupt connection services. In Windows NT, you can set these services using the DB2 Control Center, as shown in Figure 31 on page 56, and follow these steps:

- Invoke this tool, by using **Start > Programs > DB2 for Windows NT > Administration Tools > Control Center**. That will bring up the Control Center main window.
- On the left frame, expand the machine node until you find the DB2 instance that has the remote interface.
- Select the instance and right-click it to bring up a contextual pop-up menu. From the menu, choose **Configure**. This will show the Configure Instance dialog box as shown in Figure 32 on page 56.

- Select the **Communications** tab. The Control Center provides a list of supported communication protocols, along with their current parameter settings.
- Select the protocol (as an example, we selected **TCP/IP**), and update the TCP/IP Service name or use the supplied default service name.
- Click **OK**. In Windows NT, the Control Center will automatically allocate the TCP port number for the service and allocate another port for the interrupt connection service.

At that point, the database instance is ready for accepting remote connections.

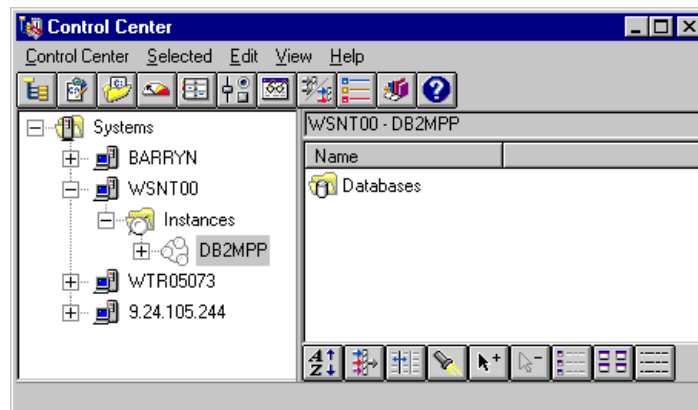


Figure 31. DB2 Control Center

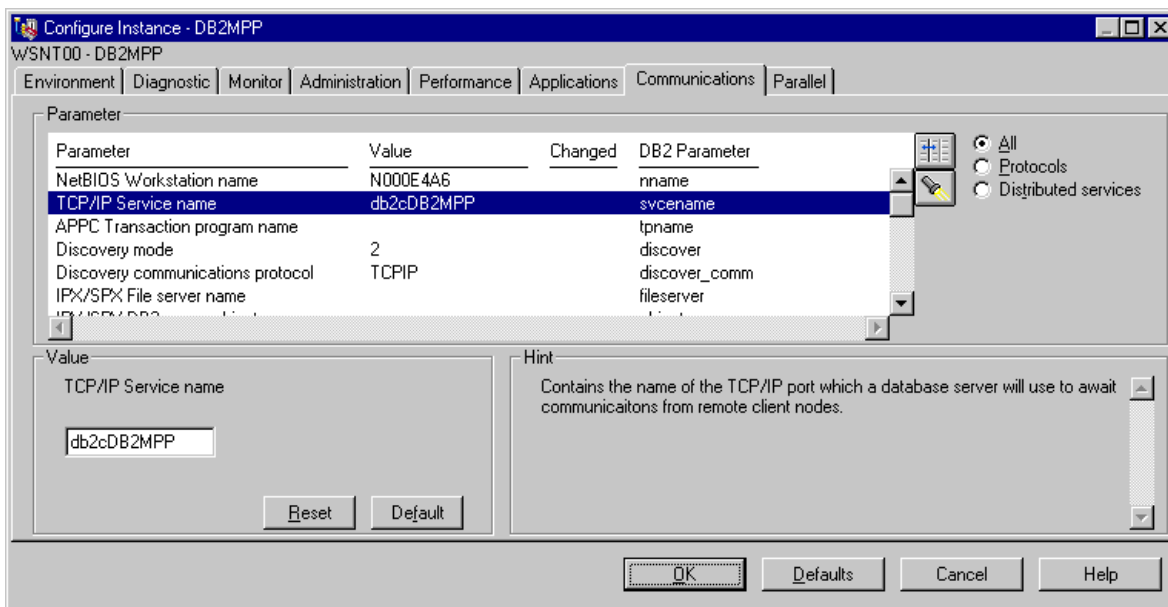


Figure 32. Configure Instance Dialog Box on DB2 Control Center

Setting Up a Remote Database Connection in DB2 Client

You can configure a DB2 Client by using the Client Configuration Assistant (CCA). You can access this tool by invoking **Start > Programs > DB2 for**

Windows NT > Client Configuration Assistant. To set up a remote database connection, follow these instructions:

- On the CCA main dialog box, click the **Add** button to call the Add Database SmartGuide window, which consists of a sequence of dialogs.
- In the first dialog box, select **Manually Configure connection to a Database.**
- The next dialog box will ask you to supply protocol parameters.
- The SmartGuide will show you a list of accessible databases in your network. Choose a database to which you want to connect.
- The last dialog box asks for a database alias name. All applications in this client machine will use that alias name to refer to the remote database.
- Click **OK** to save and activate the settings. That brings you back to the CCA main dialog box.

At that point, the database should appear in the Available DB2 Database list. To verify a connection, select a database and click the **Test** button. The system asks you for a user ID and password. Then, a message will notify you if your connection was successful.

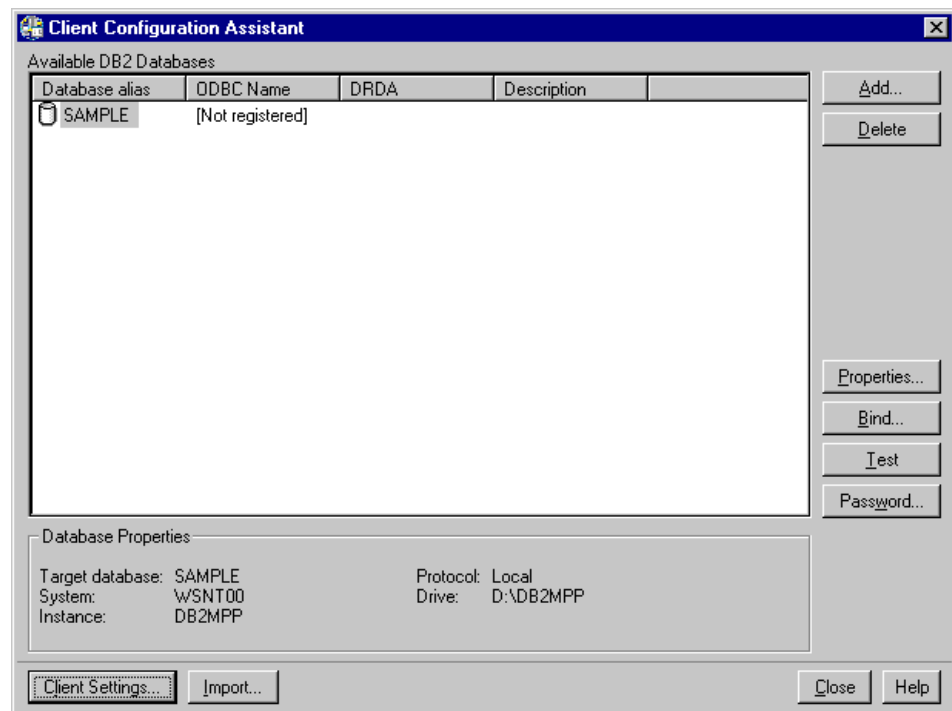


Figure 33. Client Configuration Assistant

2.1.8 Lotus Domino R5 Server for Windows NT

Lotus Domino R5 Server is a family of servers that consists of integrated messaging and Web application software and tools. There are three server products in the family: Domino Mail Server, Domino Application Server, and Domino Enterprise Server. These servers have a common architecture and can be integrated smoothly to create a wide range of applications, from messaging, collaboration, Web application, to mission-critical enterprise applications. In this section, we limit our discussion to the Web server component of the Domino

Application Server. You can find more information about other Domino products, components and features from its Web site at: <http://www.lotus.com/r5>.

Domino provides an HTTP engine (Web server) with its Application Server product. The Web server provides other Domino servers the capability to deliver services through the Web. WebSphere Application Server can utilize this Web server and make use of some of the features of Domino servers. The integration of WebSphere and Domino products is beyond the scope of this document. There is a separate project that will address this integration.

To use the HTTP engine, you can either choose to install the Domino Application Server product, or manually select the Domino Web Services component during the custom installation (Figure 34 on page 58). For detailed installation instructions, we recommend you use the Lotus Domino Yellowbook *Setting up a Domino Server*, which is available online at <http://www.notes.net/notesua.nsf>.

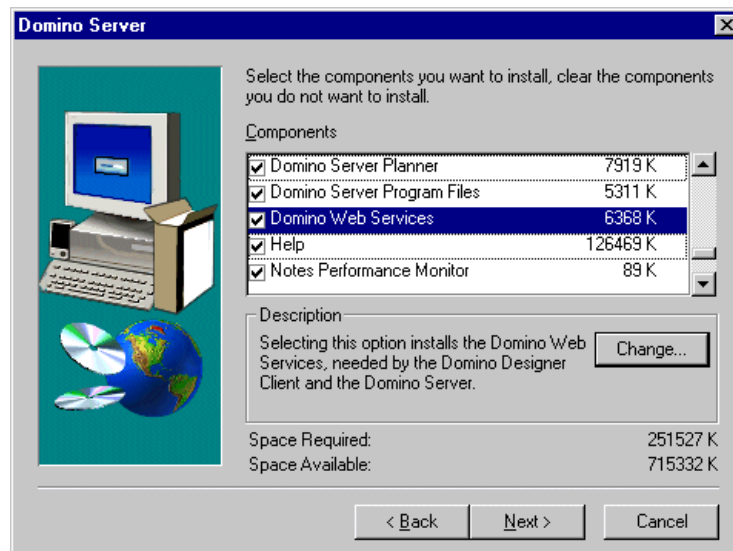


Figure 34. Selecting Domino Web Services in Domino R5 Server Installation

2.1.8.1 Setting Up a Domino Web Server on Windows NT

In the Domino environment, the Web server is delivered as a part of a Domino server. Therefore, to create a Web server, you should create a Domino server and specify the server to provide Web services. To create a Domino server, perform the following steps:

- Start Domino by using **Start > Programs > Lotus Applications > Lotus Domino Server** from the Windows NT Taskbar. If you are creating a Domino server for the first time, Domino will bring you directly to the setup.nsf application (Figure 35 on page 59). This application consists of a sequence of forms.
- In the first form for the Domino Server Setup, if you are creating a new server for a new domain, select **First Domino Server**. If you are adding a new server into an existing domain, select **Additional Domino Server**. Proceed to the next form by clicking on the ">" button.

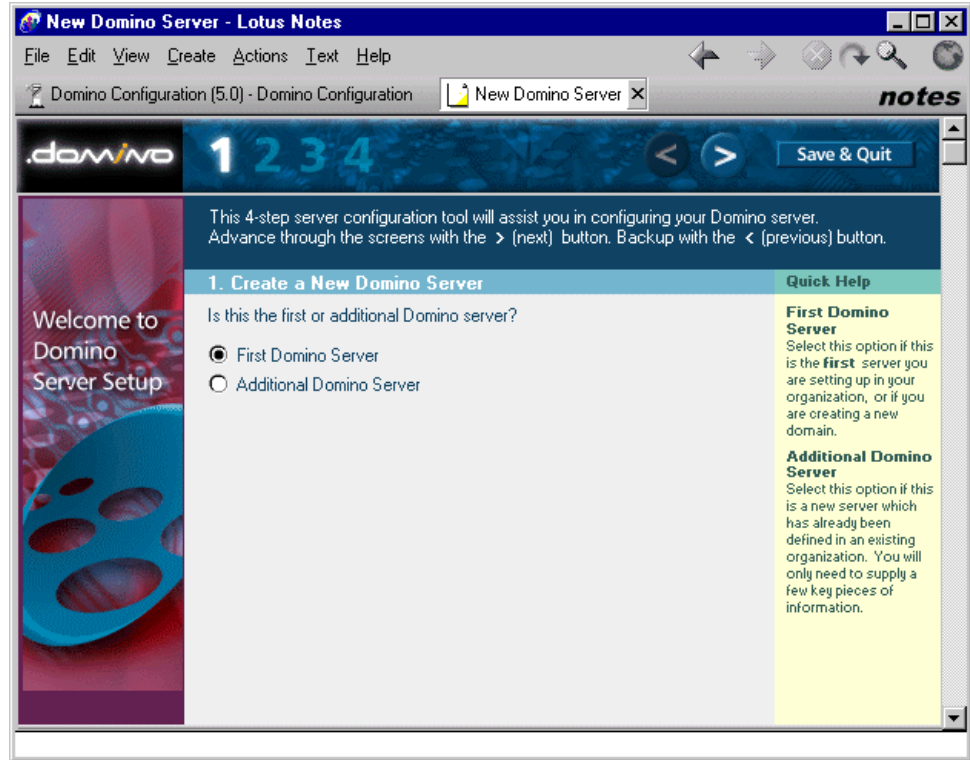


Figure 35. Creating a New Domino Server using Domino Server Setup

- In the Select a Setup Method form (Figure 36 on page 60), select **Advanced Configuration**, then go to the next form.
- In the Server Audience form (Figure 37 on page 61), choose the services that you want to provide to your client. Select the **HTTP** check box to include Web services.
- In the Administration Settings form (Figure 38 on page 62), specify a domain name, Domino server name, fully qualified domain name as Server's Hostname, as well as the server's administrator user ID and password.
- Click on **Finish** to create a new Domino server.

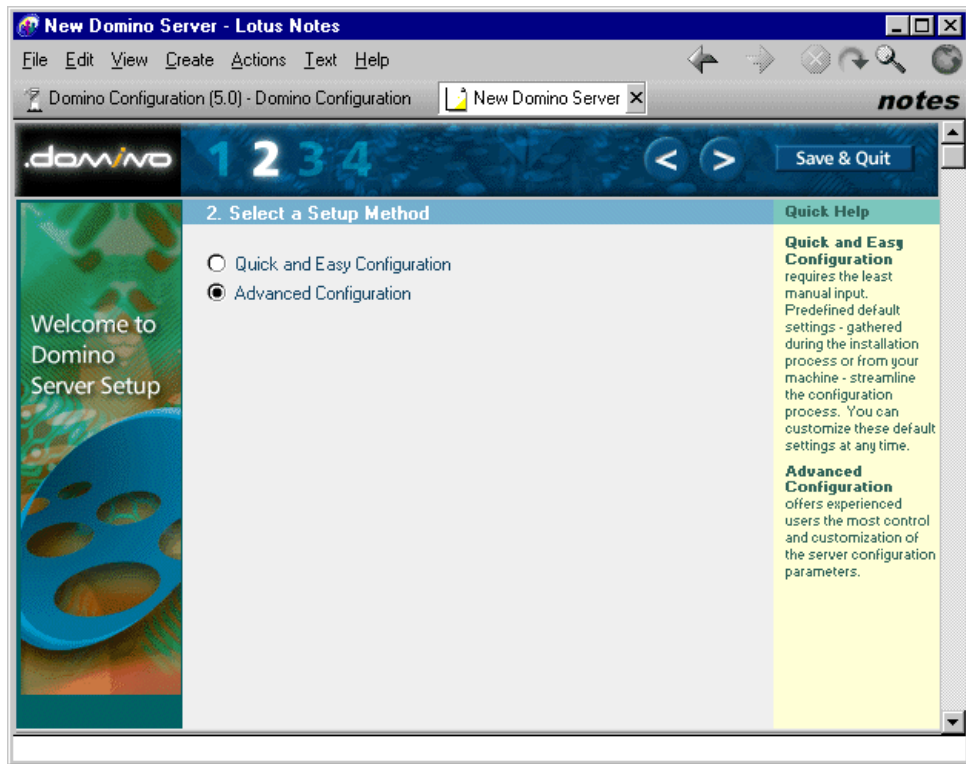


Figure 36. Select a Setup Method in Domino Server Setup

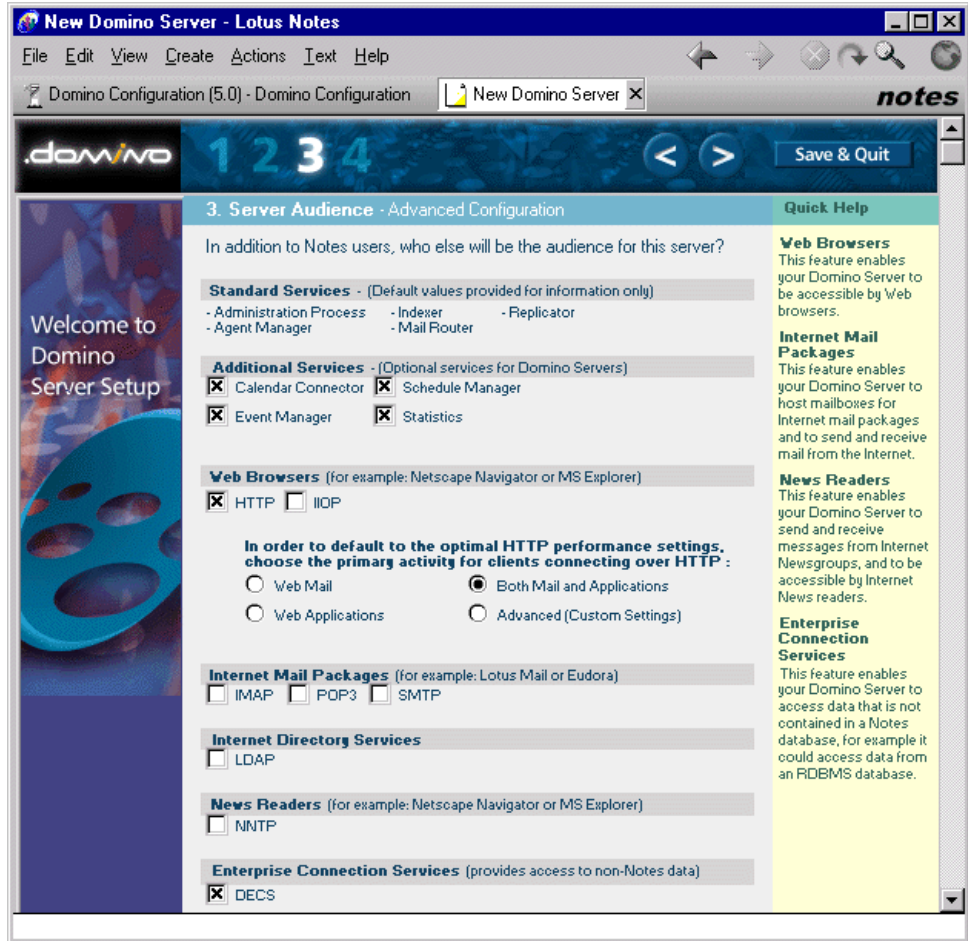


Figure 37. Server Audience in Domino Server Setup

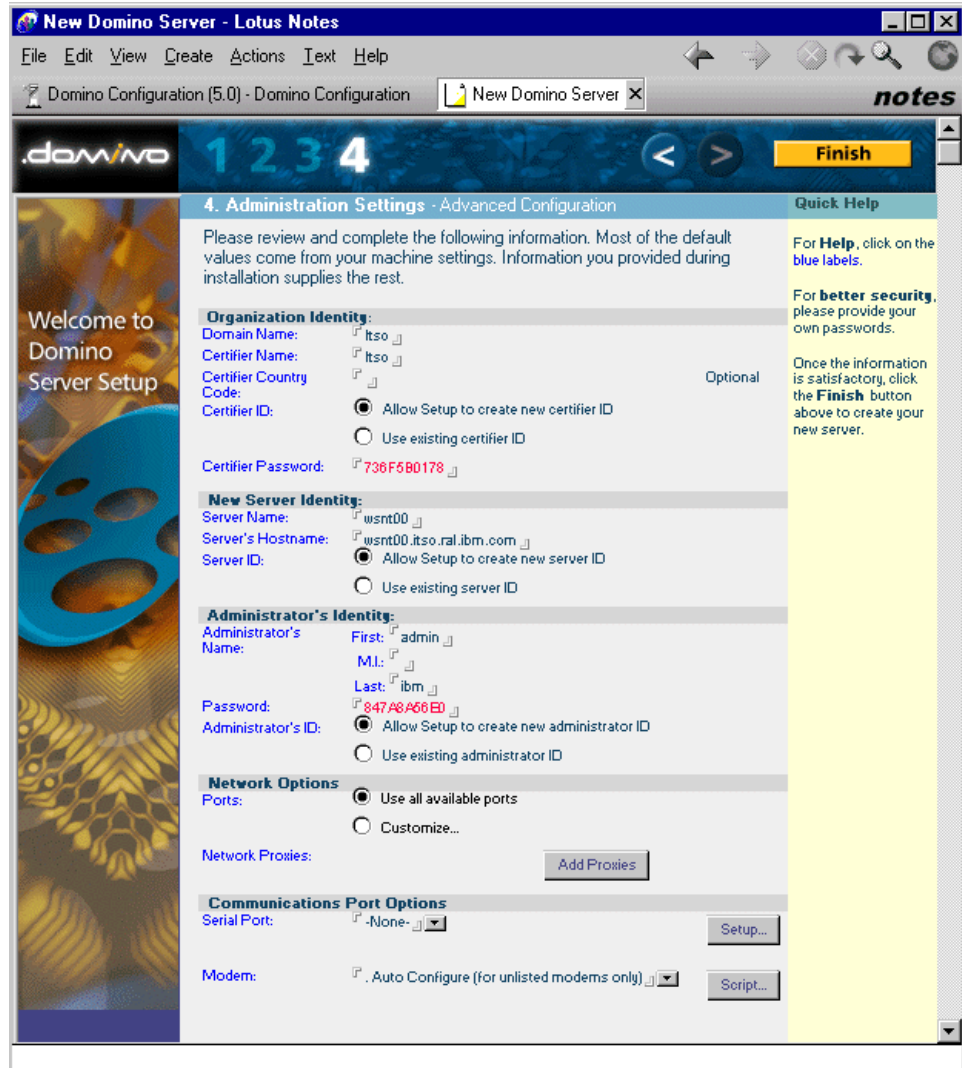


Figure 38. Administration Setting in Domino Server Setup

2.1.8.2 Domino Web Server Configuration Files

As an independent service component, the Domino Web Server has its own configuration files: `httpd.cnf` and `domino.cnf`. When the Web server is started, it generates `domino.cnf` from the Domino databases, then it reads `domino.cnf` and `httpd.cnf`. As `domino.cnf` is system generated, you should *not* modify the file. The `domino.cnf` contains Web service parameters from any Domino server service that is delivered through the Web. The `httpd.cnf` contains parameters related to general Web usage and other services outside the Domino server. You should not put Web configurations for notes applications in `httpd.cnf`.

The procedure for configuring the `httpd.cnf` is the same as configuring the Domino Go Webserver. Please see 2.1.4.1, “Configuring Domino Go Webserver” on page 37.

2.1.8.3 WebSphere Plug-In for Domino R5 Server on Windows NT

The WebSphere Application Server can deliver its services through the Domino Web Server by using a plug-in. The plug-in is `go46.dll` which is the same plug-in

that is used for Domino Go WebServer. The Web service component of Domino R5 is derived from the Domino Go WebServer.

You should install the plug-in only after you have set up the Domino Web Server. The WebSphere installation program inserts the following lines into the Domino Web server configuration file, httpd.cnf:

```
ServerInit D:\WebSphere\AppServer\plugins\nt\go46.dll:init_exit
D:\WebSphere\AppServer\properties\bootstrap.properties
ServerTerm D:\WebSphere\AppServer\plugins\nt\go46.dll:term_exit
Pass /IBMWebAS/samples/* D:\WebSphere\AppServer\samples\*
Pass /IBMWebAS/* D:\WebSphere\AppServer\web\*
```

The first line specifies the entry point for WebSphere services when the WebServer starts. The third line specifies a function, term_exit, for terminating the service. The fourth and fifth lines construct directory aliases for default WebSphere directories.

2.1.8.4 Starting the Web Server Operation

To control Web Server operation, you can use the Domino server console, which can be invoked from **Start > Program > Lotus Applications > Lotus Domino Server** or by calling `nservice.exe` from a Windows console. The console allows you to enter some commands after the `>` prompt:

```
04/22/99 11:36:52 AM Removing the version 0 free time data. Recreating it as
version 3.
04/22/99 11:36:52 AM Can't find existing Schedule database busytime.nsf,
creating a new one
04/22/99 11:36:53 AM SchedMgr: Validating Schedule Database
04/22/99 11:36:53 AM SchedMgr: Done validating Schedule Database
04/22/99 11:36:57 AM Stats agent started
04/22/99 11:36:57 AM Stats: Creating Mail-In Database record for stats
destination 'wsnt00 Stats/Itso'
04/22/99 11:36:57 AM Stats: Creating Stats Mail-In Database.
04/22/99 11:37:07 AM Creating Domino Web Administrator database...
04/22/99 11:37:08 AM DECS: Creating DECS Administrator database ...
04/22/99 11:37:12 AM Stats agent shutdown
04/22/99 11:37:13 AM Maps Extractor started
04/22/99 11:37:15 AM Setting up default monitors in Statistics & Events
database.
04/22/99 11:37:18 AM SMTP Server: Started
04/22/99 11:37:18 AM Maps Extractor: Building Maps profile
04/22/99 11:37:18 AM Maps Extractor: Maps profile built OK
04/22/99 11:37:18 AM DECS: DECS Administrator database created
04/22/99 11:37:21 AM DECS Server started
04/22/99 11:37:23 AM Domino Web Administrator database created
04/22/99 11:37:28 AM Database Server started
04/22/99 11:37:35 AM HTTP Web Server started
>
```

2.2 WebSphere Installation on Windows NT

To ensure a smooth installation process, before you install the WebSphere in your machine, perform the following:

- Choose an edition of WebSphere Application Server that matches your requirements. See 1.1.1, “WebSphere Application Server” on page 2 for a comparison of the features of the different WebSphere editions.
- Make sure that all required hardware and software requirements are set up and running in your machine. This includes a working JDK/JRE, Web browser, Web server, and database system.
- You should set up TCP/IP properties such as Host Name, Domain Name, IP, gateway and DNS addresses. Ensure that your Internet/intranet environment works.
- Set up a local Administrator user ID to run Windows NT services. We recommend you restrict access to the machine. You should follow your company’s system security procedures.
- To minimize the risk for corrupting shared files, shut down the Web server on which you want to install the WebSphere. Stop any running Windows applications.

The installation process consists of a sequence of dialog boxes. After each dialog box, you can proceed to the next dialog box by clicking the **Next>** button. If you miss some information in a screen, you can always go back by clicking the **Back>** button.

- The first installation dialog box (Figure 39 on page 65) warns you to stop any running Windows program and Web server in your machine.
- In the next dialog box (Figure 40 on page 65), you have to specify a directory to install. This directory will be the WebSphere Application Server root directory.
- The next dialog box (Figure 41 on page 66) asks you to select the application server components. Select any component that you need.
- The following dialog box (Figure 42 on page 66) lists the JDK or JRE installed in your system. The installation program obtains this information from the Windows NT registry. Choose a JDK/JRE system from the list. Choosing a JRE only will prevent developers from working on the system. If you want an alternative JDK/JRE, specify its directory.
- Choose an application server plug-in in the next dialog box (Figure 43 on page 67). The installation program will detect any installed Web server from the registry. By default, selections for Apache and IBM HTTP servers are enabled. If you have installed other Web server products, the dialog box will enable selections on the corresponding product. If your Web server has a later version than those listed in the dialog box, choose the latest plug-in version available.
- Specify a folder to put the WebSphere programs. After that, click **Next>** to start copying files.
- The installation program will take some time to install on your system.
- After it finishes, the installation program may ask you for the location of the Web server configuration file (Figure 44 on page 67). For Domino Go Webserver or Lotus Domino, the file is httpd.cnf; for Netscape Server the file is obj.conf. IIS stores the configuration in the registry. For Apache or IBM HTTP Server the file is httpd.conf.

- In the last window you can read the readme.txt file or click **Finish** to complete the installation.
- Reboot your computer, and WebSphere should be ready after the system is up.

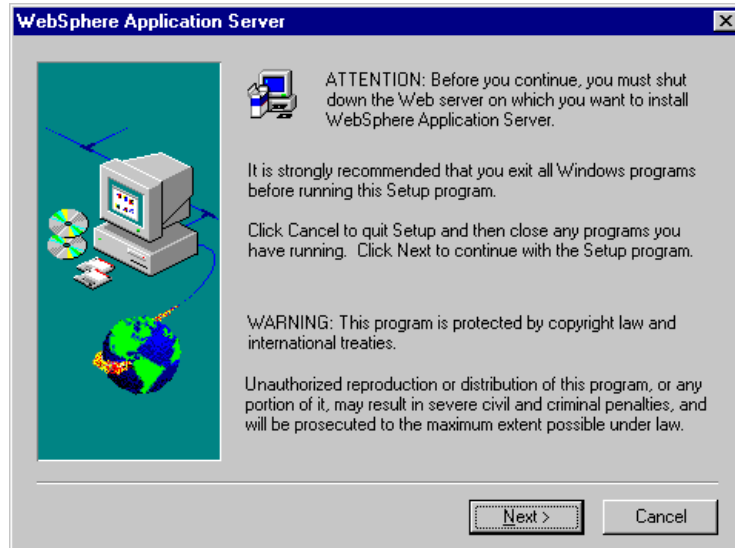


Figure 39. WebSphere Application Server on Windows NT Installation Welcome

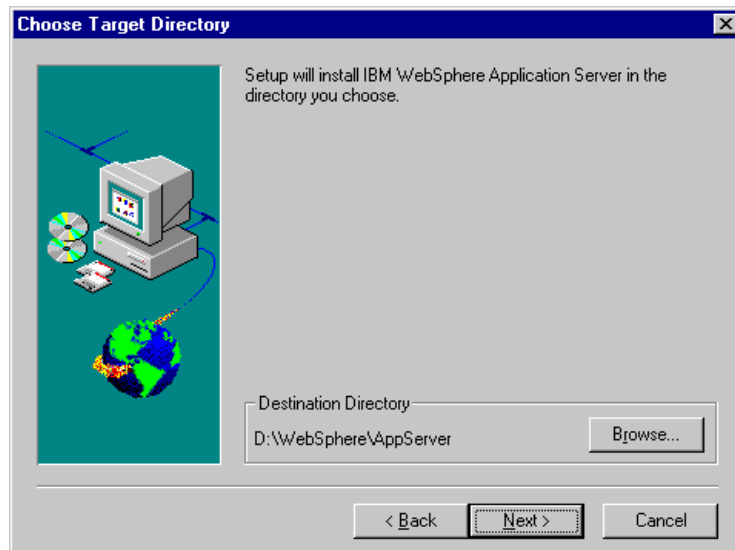


Figure 40. WebSphere Installation on Windows NT Target Directory

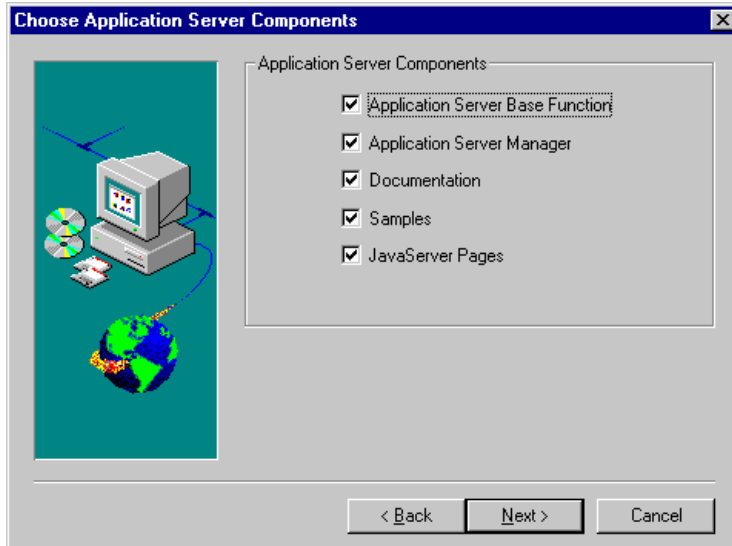


Figure 41. WebSphere Installation on Windows NT - Choose Application Server Components

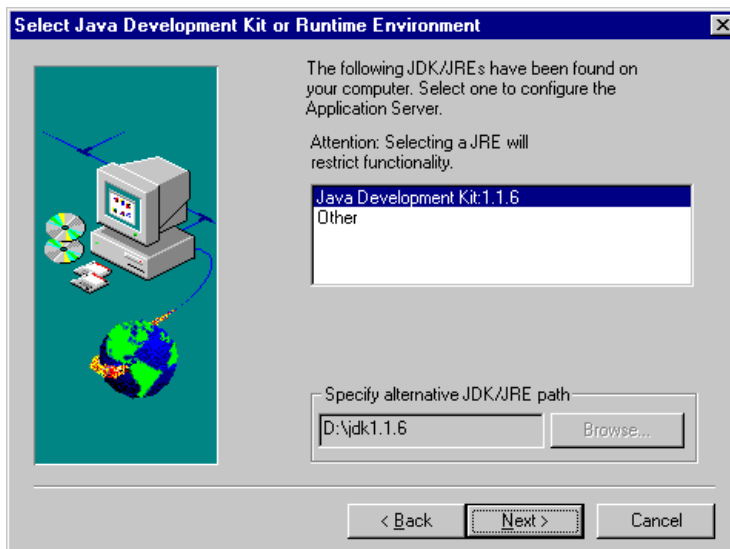


Figure 42. Select JDK or JRE

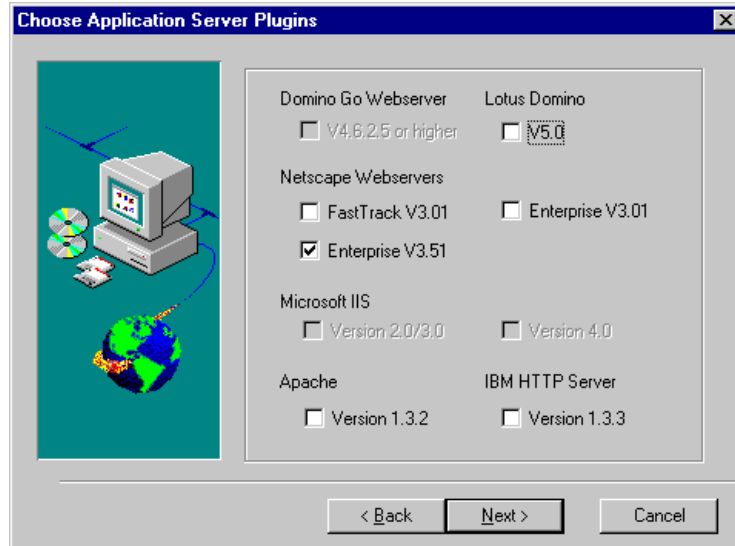


Figure 43. Select Application Server Plug-In

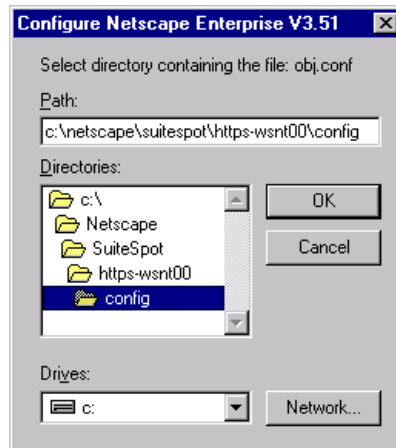


Figure 44. WebSphere Installation on Windows NT - Specifying Web Server Configuration File

2.3 Infrastructure Installation for AIX V4.3.2

The following products were installed on AIX V4.3.2:

- JDK
- HTTP Server
- Domino Go WebServer
- Netscape SuiteSpot
- DB2 UDB
- Domino R5

2.3.1 JDK 1.1.6 Installation and Setup Procedure

This section will show you how to install a JDK on an AIX machine. You can obtain the JDK for AIX from the WebSphere installation CD or by downloading it

from <http://www.ibm.com/java/jdk>. On the AIX platform, the JDK is available in tar format. You need to uncompress it into a temporary directory before starting the installation.

To perform the installation, you need to be a user that has root privileges. In an AIX terminal window, enter the following command:

```
smitty installp
```

This will bring up the AIX System Management Interface Tool window and put you at the Install and Update Software menu as shown in Figure 45 on page 68:

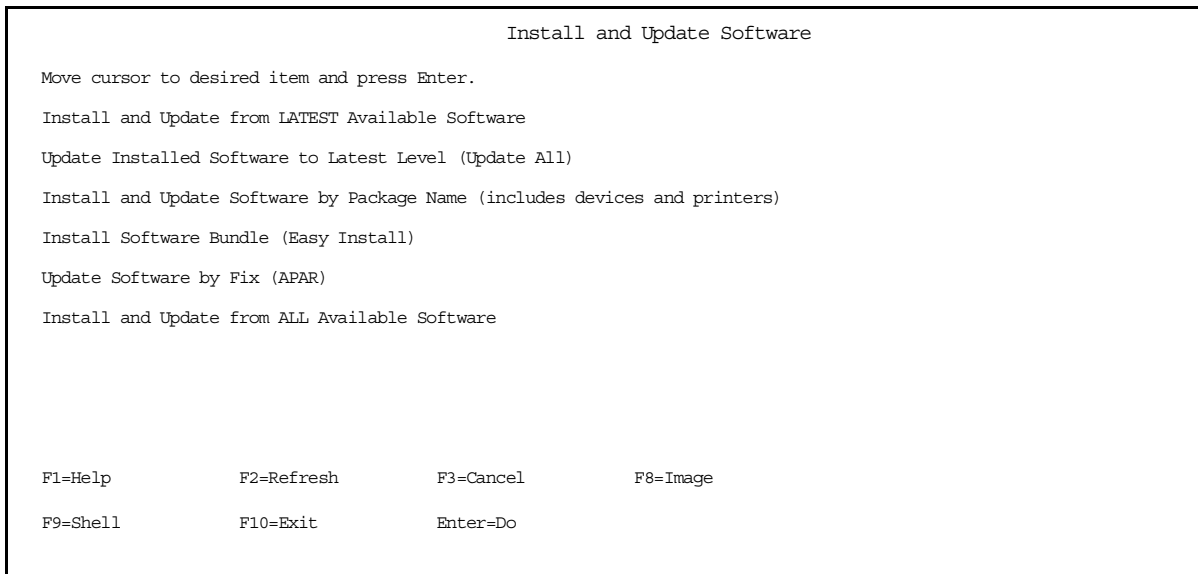


Figure 45. JDK Installation on AIX - Install and Update Software Menu on smitty

Choose **Install and Update from LATEST Available Software** to invoke the installation process. After that, enter the device or directory where the JDK installation package is located, as shown in Figure 46 on page 69.


```

                                Install and Update from LATEST Available Software

Type or select a value for the entry field.
Press Enter AFTER making all desired changes.
* INPUT device / directory for software          [/99swa109/JDK]      +

F1=Help          F2=Refresh          F3=Cancel          F4=List
F5=Reset          F6=Command          F7=Edit            F8=Image
F9=Shell          F10=Exit            Enter=Do

```

Figure 46. JDK Installation on AIX -Specifying Installation Source Directory

smitty will then let you choose which software components to install, as shown in Figure 47 on page 69. Choose **_all_latest** and click **Enter** to start the installation.

```

                                Install and Update from LATEST Available Software

Type or select values in entry fields.
Press Enter AFTER making all desired changes.
* INPUT device / directory for software          /99swa109/JDK
* SOFTWARE to install                          [_all_latest]      +
PREVIEW only? (install operation will NOT occur)  no                +
COMMIT software updates?                       yes               +
SAVE replaced files?                           no                +
AUTOMATICALLY install requisite software?       yes               +
EXTEND file systems if space needed?            yes               +
OVERWRITE same or newer versions?              no                +
VERIFY install and check file sizes?           no                +
Include corresponding LANGUAGE filesets?        yes               +
DETAILED output?                               no                +
Process multiple volumes?                      yes               +

F1=Help          F2=Refresh          F3=Cancel          F4=List
F5=Reset          F6=Command          F7=Edit            F8=Image
F9=Shell          F10=Exit            Enter=Do

```

Figure 47. JDK Installation on AIX - Confirmation before Starting the Installation

After it finishes, it will provide you with an installation summary at the bottom of the command status screen, as shown in Figure 48 on page 70. You should verify each component's installation status.

```

                                COMMAND STATUS
Command: OK          stdout: yes      stderr: no
Before command completion, additional instructions may appear below.
[MORE...144]
Installation Summary
-----
Name                  Level          Part          Event          Result
-----
Java.rte.Dt           1.1.6.0       USR           APPLY          SUCCESS
Java.rte.Dt           1.1.6.0       ROOT         APPLY          SUCCESS
Java.adt.src          1.1.6.0       USR           APPLY          SUCCESS
Java.adt.includes     1.1.6.0       USR           APPLY          SUCCESS
Java.adt.docs         1.1.6.0       USR           APPLY          SUCCESS
Java.adt.src          1.1.6.1       USR           APPLY          SUCCESS
Java.adt.src          1.1.6.1       USR           COMMIT         SUCCESS
[BOTTOM]
F1=Help              F2=Refresh     F3=Cancel     F6=Command
F8=Image             F9=Shell       F10=Exit      /=Find
n=Find Next

```

Figure 48. JDK Installation on AIX -Installation Summary

Setting Environment Variable

After installation, the JDK will know the path to its own class libraries. To enable the JDK to find other class libraries, you should set the system CLASSPATH environment variable. You should also set the JAVA_HOME environment variable to point to the base directory for the JDK, which is /usr/jdk_base by default.

You can set this variable in your current AIX session by using the export command or put these commands into your login profile.

```

export CLASSPATH=$CLASSPATH:./path1:path2:any path to class library
export JAVA_HOME=/usr/jdk_base

```

2.3.2 IBM HTTP Server V1.3.3 for AIX

For the AIX platform, you can obtain the IBM HTTP Server from the WebSphere CD or obtain it in tar format from:

<http://www.software.ibm.com/webservers/httpservers/download.html>.

2.3.2.1 Installing IBM HTTPServer

To install the HTTP Server using the WebSphere installation CD, you should select the IBM HTTP Server, the IBM HTTP Server plug-in as a component and a sub-component to install. The WebSphere installation program will silently install the HTTP Server and automatically modify the HTTP Server configuration file.

If you install the HTTP Server from a downloaded package, you should install the HTTP Server first before installing WebSphere. The WebSphere installation

program will insert plug-in information into the HTTP Server configuration file. To install the HTTP Server separately, you should log on as a root user, and perform the following steps:

- Uncompress the downloaded tar file using:

```
tar -xvf <tar-filename>
```

- Change directory to the directory that contains uncompressed results. If the installation package table of contents (the .toc file) does not exist, create it with the following command while still in the install directory:

```
inutoc .
```

- Call the AIX smitty utility for package installation:

```
smitty installp
```

- Specify the source directory of installation package, and click **Enter** to start the installation. At the end of the installation process, you can verify the installation status from the installation summary.

2.3.2.2 Configuring IBM HTTP Server

After the HTTP Server is installed, the basic tasks for configuring a Web Server are:

- Specify your domain name and Web server port.
- Specify a directory alias.
- Specify your access control.

You can set up the Web server by editing the HTTP Server configuration file, `httpd.conf`, which is located in `/usr/lpp/HTTPServer/etc` directory. The process for modifying this file is the same as the HTTP Server setup in Windows NT platform. Section 2.1.3.1, “Configuring IBM HTTP Server” on page 34 explains some basic directives to control Web server behavior. You can refer back to this section for configuring a Web server.

2.3.2.3 Controlling Web Server Operation

The HTTP Server provides a utility program, called `apachectl`, to control basic Web server operations. The program is located in the `/usr/lpp/HTTPServer/sbin` directory. The syntax for this utility is `apachectl <command>`. The `<command>` argument can be: `start`, `stop`, `restart`, `status`, or `configtest`. As an example, the following command starts the Web server:

```
apachectl start
```

To stop the Web server:

```
apachectl stop
```

2.3.2.4 WebSphere Plug-In for HTTPServer on AIX

Basically, the HTTP Server is an Apache Web server, which is built upon several base modules. Each module provides a specific service. The WebSphere Application Server delivers its service to the HTTP Server using a module (or plug-in), named `ibm_app_server_module`. This module is implemented as an object file, `mod_ibm_app_server.so`. When the HTTP Server starts, it loads and starts the module. To enable the HTTP Server to recognize the module, WebSphere inserts the following lines into the `httpd.conf` file:

```
LoadModule ibm_app_server_module
/usr/WebSphere/AppServer/plugins/aix/mod_ibm_app_server.so
```

WebSphere also inserts several directives, as listed below, to be passed into `ibm_app_server_module` as initialization parameters:

```
NcfAppServerConfig BootFile
    /usr/WebSphere/AppServer/properties/bootstrap.properties
NcfAppServerConfig LogFile    /usr/WebSphere/AppServer/logs/apache.log
NcfAppServerConfig LogLevel  TRACE|INFORM|ERROR
```

2.3.3 Domino Go Webserver

The Domino Go Webserver is a Web server product from Lotus. It is also known as the IBM Internet Connection Server. The product is available on many platforms, including Windows NT, AIX, Solaris, HP/UX, OS/2 and OS/390. Lotus has integrated the Web server into its Lotus Domino R5 server family. It is the Web Services component in the Domino R5 Application Server product. On the AIX platform, you can obtain the Domino Go Webserver as a separate installation package from IBM.

2.3.3.1 Configuring Domino Go Webserver

Domino Go Webserver provides Web-based configuration and administration forms. You can access these online forms by accessing `http://<your.domain.name>/admin-bin/cfgin/initial`, as shown in Figure 19 on page 38. These forms are easy to use with a complete description as well as examples for each parameter. As an alternative, you may configure the Web server by using its configuration file which is `httpd.cnf`.

The procedure for configuring the Web server on AIX is the same as on the Windows NT platform. Refer to 2.1.4.1, “Configuring Domino Go Webserver” on page 37, for a list of basic configuration directives and the steps for setting up the Web server.

2.3.3.2 WebSphere Plug-In for Domino Go Webserver on AIX

WebSphere can run on top of Domino Go Webserver by using a plug-in. For Domino Go Webserver 4.x or later on AIX platform, the plug-in file is `libdomino.so`.

During installation, the WebSphere installation program adds several entries into the `httpd.cnf` file:

```
ServerInit /usr/WebSphere/AppServer/plugins/aix/libdomino.so:init_exit
    /usr/WebSphere/AppServer/properties/bootstrap.properties
ServerTerm /usr/WebSphere/AppServer/plugins/aix/libdomino.so:term_exit
Pass /IBMWebAS/samples/* /usr/WebSphere/AppServer/samples/*
Pass /IBMWebAS/* /usr/WebSphere/AppServer/web/*
```

The first and second entries map server initialization and termination functions into a corresponding API entry. The other entries are Websphere default mapping rules.

2.3.4 Netscape SuiteSpot V3.X for AIX

The Netscape SuiteSpot for AIX is a family of server products that includes Web server, directory server, proxy server, media server, and several other server

types. The Web server product in the SuiteSpot family is the Netscape Enterprise Server. In this section, we limit our discussion to the Enterprise Server.

SuiteSpot provides an Administration Server to configure any server in the family. The Netscape SuiteSpot for AIX Administration Server is a Web service tool used for configuring SuiteSpot servers. The Administration Server for AIX has the same functions, look and feel as the Administration Server for Windows NT. The process for setting up and configuring the Web server is the same as those explained for the Windows NT platform, except for UNIX directory naming conventions. See 2.1.5, "Netscape SuiteSpot V3.x for Windows NT" on page 41.

2.3.4.1 WebSphere Netscape Enterprise for AIX Plug-Ins

The WebSphere Application Server provides application services through a Netscape Enterprise plug-in. The plug-in is loaded when the WebServer starts. The Netscape Enterprise configuration file, `obj.conf`, specifies the name and location of plug-ins. WebSphere's installation program modifies this configuration file. Since the configuration file is only available after you set up an Enterprise Web server, you should install WebSphere *after* setting up the Web server. In the WebSphere installation process, the installation program will ask you to select the appropriate Web server plug-in and ask you for the location of the configuration file.

In Netscape Enterprise for AIX, the configuration file is:

```
<SuiteSpotRoot>/https-<ServerIdentifier>/config/obj.conf
```

and the plug-in module is `libnsXX.so`, where `XX` is the Enterprise version number.

The WebSphere installation program will automatically insert the following lines inside the `obj.conf` file:

```
Init fn="load-modules" funcs="init_exit,service_exit,term_exit"  
shlib="/usr/WebSphere/AppServer/plugins/aix/libns35.so"  
  
Init fn="init_exit"  
bootstrap.properties="/usr/WebSphere/AppServer/properties/bootstrap.properties"  
"  
  
NameTrans from="/IBMWebAS/samples" fn="pfx2dir"  
dir="/usr/WebSphere/AppServer/samples"  
  
NameTrans from="/IBMWebAS" fn="pfx2dir" dir="/usr/WebSphere/AppServer/web"
```

The first line indicates that in the Web server initialization, the Web server should load the WebSphere plug-in module, `libns35.so`. The second line tells the Web server to invoke `init_exit` for starting the module using specified bootstrap properties. The third and fourth lines specify directory aliases for WebSphere default document directories.

After the WebSphere installation, you should apply the changes in `obj.conf` into the Web server. You can do this by clicking the **Apply** button on any Server Manager form for the corresponding Web server.

2.3.5 DB2 Universal Database (UDB) for AIX

DB2 UDB products for AIX provide the same functions as DB2 UDB for Windows NT platform. See 2.1.7, “DB2 Universal Database (UDB) for Windows NT” on page 52 for a brief introduction of DB2 products and components.

DB2 UDB comes with full Java support. It provides JDBC and SQLJ drivers for a smooth integration with the WebSphere application server. Section 6.2, “Using DB2 UDB for WebSphere Applications” on page 349 describes techniques for accessing the DB2 database.

DB2 UDB for AIX product packaging is the same as DB2 UDB on Windows NT. If you install DB2 UDB and WebSphere on the same machine, you do not need to install DB2 CAE. If DB2 UDB and WebSphere are on different machines, you need to install DB2 CAE or SDK on the WebSphere machine and configure a remote interface to the DB2 UDB server.

2.3.5.1 Setting Up DB2 in the WebSphere Environment

In the WebSphere environment, you should install the DB2 Client and DB2 Java support components. Both components are available in DB2 UDB, CAE and SDK products. Make sure that you select Java support in the DB2 setup program.

Setting Up Java Environments

To use DB2 in the WebSphere environment, you should add the DB2 Java class paths and native implementation library path into the system environment and WebSphere Java engine setup. The Java path information in the system environment is required for the DB2 Client component or for developing applications.

Add the following entries into the system CLASSPATH:

```
CLASSPATH=$CLASSPATH:${INSTHOME}/sqllib/java/db2java.zip
CLASSPATH=$CLASSPATH:${INSTHOME}/sqllib/java/runtime.zip
CLASSPATH=$CLASSPATH:${INSTHOME}/sqllib/java/sqlj.zip
export CLASSPATH
```

```
LD_LIBRARY_PATH=$(LD_LIBRARY_PATH):${INSTHOME}/sqllib/lib
export LD_LIBRARY_PATH
```

where INSTHOME is the DB2 instance home directory.

To add the path information into the WebSphere Java engine setup, follow these steps:

- Log on to the WebSphere Administration tool as an administrator.
- On the left frame, select **Setup > Java Engine**. This will bring up the Java Engine setup dialog box as shown in Figure 30 on page 55.
- On the Paths tab, append the following entries to the Application Server Classpath:

```
D:\SQLLIB\java\db2.zip;D:\SQLLIB\java\sqlj.zip;D:\SQLLIB\java\runtime.zip
```

- Append the following entry to the User Libpath:

```
D:\SQLLIB\bin
```

- Click **Save** and log off from the Administration tool.

- Stop the Web server and kill all WebSphere processes. You can get the WebServer process identifier by using:

```
ps -ef | grep WebSphere
```

- Restart the Web server to activate the new settings.

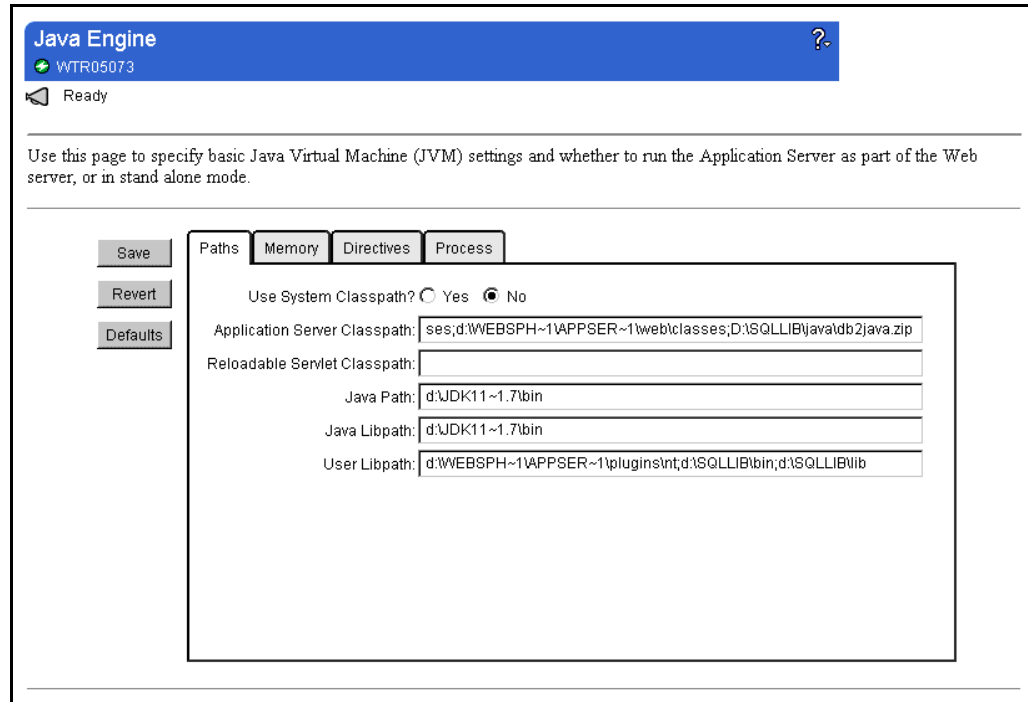


Figure 49. Setting Up DB2 Java Path in WebSphere Administration Tool

2.3.5.2 Configuring Remote Interface

A remote interface is required if the application server and the database server are not in the same machine. The remote interface links the DB2 Client component to a remote database. Configuring a remote interface involves setting up server communication profiles and defining a remote database connection in the client machine.

Setting Up Server Communication Profile

DB2 UDB for AIX includes DB2 Communication Support components to enable clients, either a local client component or a remote one, to communicate with the DB2 Server. This communication component can support multiple communication protocols concurrently. The protocol can be any combination of TCP/IP, APPC or IPXSPX. You should have a System Administrative Authority (SYSADM) to set up this service.

To identify which protocol to use, the system uses the DB2COMM environment variable. To set DB2COMM enter the following command:

```
db2setup DB2COMM=TCP/IP
db2stop
db2start
```

After restarting, DB2 will have a new DB2COMM value.

The remote client interface requires two communication support services: one for the connection service, and the other for the interrupt connection services. The service parameters (name and addresses) are set according to the underlying protocol service and address notation.

For TCP/IP communication services, there are two TCP/IP services that should be set in the TCP/IP service configuration file: /etc/services file. Their port numbering has the following restriction: if the port number for the connection service is n , then the port number for interrupt connection service is $n+1$.

For example, to create conserv TCP/IP service on port 10000, put the following lines into /etc/services file:

```
conserv 10000/tcp # DB2 interrupt connection service port
conservi 10001/tcp # DB2 interrupt connection service port
```

After these services are created, you need to make DB2 recognize these services. You can do it by updating the database manager configuration with the new connection service name. The system will automatically identify the interrupt connection service by its port number:

```
db2 update database manager configuration using svcname <servicename>
db2stop
db2start
```

After restarting the database system, you can verify configuration values by entering the following command:

```
db2 get database manager configuration
```

It will show the database manager configuration and it should contain the following line:

```
...
TCP/IP Service name (SVCENAME) = <servicename>
...
```

Setting Up Remote Database Connection at DB2 Client

Setting DB2 client components is done by specifying a host address, indicating service names involved and cataloging the TCP/IP node and database.

To specify the host address, create an entry in /etc/hosts to enable the system to resolve host name into IP address. For example, insert the following line into /etc/hosts:

```
9.24.104.68 hostnt00 #host address for serverhost
```

To specify the service offered by the server, the system must resolve the service name into a port number. This is done by creating a new entry in the /etc/services file:

```
conservice 10000/tcp # DB2 connection service port
```

To describe the remote node, you should catalog the node by issuing the catalog command to DB2. For example, to catalog node nodent00 on remote host hostnt00 using service name conservice with the TCP/IP protocol, enter the following commands:


```
catalog tcpip node nodent00 remote hostnt00 server conservice
terminate
```

Before a client can connect to a remote database, the client should catalog the database. For example, to catalog the database *sample* at node *nodent00*, and identify it locally as *sample1*, enter the following command:

```
catalog database sample as sample1 at node nodent00
terminate
```

2.3.6 Lotus Domino R5 Server for AIX

A brief introduction to Lotus Domino R5 is presented in 2.1.8, “Lotus Domino R5 Server for Windows NT” on page 57. Since we are going to work only with the Web server installation for WebSphere, we only described setting up a Domino Web server.

2.3.6.1 Setting Up a Domino Web Server on AIX

Creating a Web server on the Domino platform involved creating a Domino server and specifying that an HTTP service is to be included in the Server. The server setup application, *setupweb.nsf*, is a Web-based tool. To invoke the tool, perform the following steps:

- Log in as a Notes user that owns the Domino data directory and Domino server process. That user ID was set up during the Domino installation.
- Change the directory to the Domino data directory, for example, `/usr/lotus/notesdata`.
- Start the HTTP engine and ask it to load the Domino Server Setup service by giving `httpsetup` as the argument:

```
/usr/lotus/bin/http httpsetup
```

This will start the Domino Server Setup server, which is a Web-based application located on port 8081.

- Open a Web browser and go to your domain site on port 8081. For example, if the domain name is `rs600031e.itso.ral.ibm.com`, enter the following line into Web browser location entry field:

```
http://rs600031e.itso.ral.ibm.com:8081
```

This will invoke Domino Server Setup in your Web browser as shown in Figure 50 on page 78.

At that point, the application has the same look and feel as the Domino Server Setup on Windows NT. Follow the steps in 2.1.8.1, “Setting Up a Domino Web Server on Windows NT” on page 58.

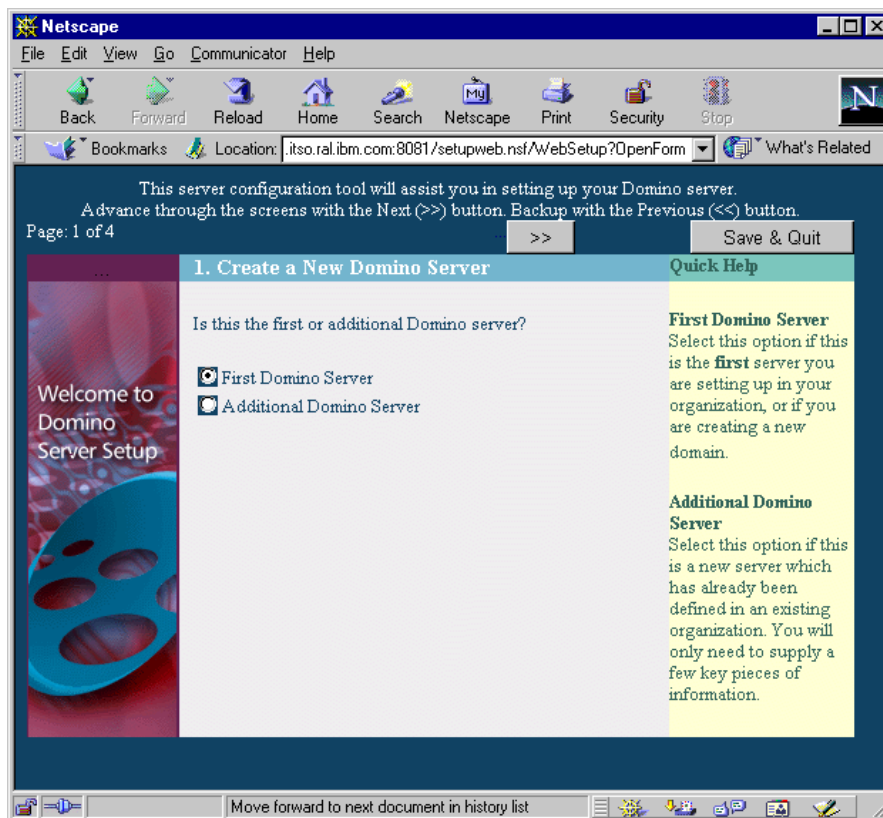


Figure 50. Domino Server Setup in AIX Platform is a Web Application

2.3.6.2 Domino Web Server Configuration Files

As an independent service component, the Domino Web Server has its own configuration files: `httpd.cnf` and `domino.cnf`. When the Web server is started, it generates `domino.cnf` from `setup.nsf` and other Domino databases. It then reads `domino.cnf` and `httpd.cnf`. As `domino.cnf` is system generated, you should not manually modify the file. The `domino.cnf` file contains Web service parameters from any Domino server service that is delivered through the Web. The `httpd.cnf` configures parameters related to general Web usage and other services outside the Domino server. You should not put Web configuration for notes applications in `httpd.cnf`.

The procedure for configuring the `httpd.cnf` is the same as configuring in Domino Go Webserver. See 2.1.4.1, “Configuring Domino Go Webserver” on page 37.

2.3.6.3 WebSphere Plug-In for Domino R5 Server on AIX

The WebSphere Application Server can use the Domino Web Server to deliver its services by using a plug-in. The plug-in is implemented as an object file, `libdomino.so`. In the WebSphere installation, the installation program inserts the following lines into `httpd.cnf`:

```
ServerInit /usr/WebSphere/AppServer/plugins/aix/libdomino.so:init_exit
           /usr/WebSphere/AppServer/properties/bootstrap.properties
ServerTerm /usr/WebSphere/AppServer/plugins/aix/libdomino.so:term_exit
Pass /IBMWebAS/samples/* /usr/WebSphere/AppServer/samples/*
Pass /IBMWebAS/* /usr/WebSphere/AppServer/web/*
```

When the Web server starts, it reads the httpd.cnf configuration file. The ServerInit directive instructs the Web Server to load the plug-in. The Web Server also reads additional Pass directives to create directory aliases for the default WebSphere directory.

2.3.6.4 Controlling Web Server Operation

To control Web server operation, log in as a Notes user, and call the server from an AIX terminal window:

```
$ server

Lotus Domino (r) Server, Release 5.0 (Intl), 30 March 1999
Copyright (c) 1985-1999, Lotus Development Corporation, All Rights Reserved

04/22/99 07:10:21 PM Mail Router started for domain ITSO
04/22/99 07:10:21 PM Router: Internet SMTP host rs600031e in domain
itso.ral.ibm.com
04/22/99 07:10:25 PM Database Replicator started
04/22/99 07:10:30 PM Index update process started
04/22/99 07:10:36 PM Agent Manager started
04/22/99 07:10:37 PM JVM: Java Virtual Machine initialized.
04/22/99 07:10:37 PM AMgr: Executive '1' started
04/22/99 07:10:41 PM rs600031e/Itso/us is the Administration Server of the
Domino Directory.
04/22/99 07:10:41 PM Administration Process started
04/22/99 07:10:46 PM Calendar Connector started
04/22/99 07:10:51 PM Schedule Manager started
04/22/99 07:10:51 PM SchedMgr: Validating Schedule Database
04/22/99 07:10:51 PM SchedMgr: Done validating Schedule Database
04/22/99 07:10:56 PM Event Dispatcher started
04/22/99 07:11:01 PM Stats agent started
04/22/99 07:11:08 PM HTTP Web Server started
04/22/99 07:11:11 PM DECS Server started
04/22/99 07:11:16 PM Maps Extractor started
04/22/99 07:11:20 PM Database Server started
>
```

2.4 WebSphere Installation on AIX

To ensure a smooth installation process, before you install the WebSphere in your AIX machine, do the following:

- Choose an edition of WebSphere Application Server that matches your requirements. You can refer to 1.1.1, “WebSphere Application Server” on page 2 to compare features of WebSphere editions.
- Make sure that all hardware and software requirements are set up and running on your machine. This includes working versions of JDK/JRE, Web browser, WebServer, and database systems.
- Ensure that you have configured your TCP/IP environment correctly. Verify your machine’s Host Name, Domain Name, IP, routing and DNS addresses. Make sure that your Internet/intranet access from/to your machine is working properly.
- You should have an AIX user ID for running the Web Server. WebSphere will use the same user ID for executing its process. To install WebSphere Application Server, you can either log in as a root or a user ID that can run the Web Server.

The installation process consists of a sequence of dialog boxes. After each dialog box, you can proceed to the next dialog box by clicking the **Next>** button. If you miss some information in a screen, you can always go back by clicking the **Back>** button.

- After the Welcome dialog box (Figure 51 on page 80), go to the next dialog box.
- Specify a directory to install the WebSphere (Figure 52 on page 81). This directory will be the WebSphere Application Server root directory.
- The next dialog box (Figure 53 on page 81) asks you to select the application server components that you want to install. To select the Web Server on which WebSphere will run, select **Application Server** from the list on the left and select the appropriate Web server plug-ins on the right. If your Web server has a later version than those listed in the dialog box, choose the latest plug-in version available.
- The Installation program will ask you for the location of your Web server configuration file (Figure 44 on page 67). For Domino Go Webserver or Lotus Domino the file is httpd.cnf; for Netscape Server the file is obj.conf; for Apache or IBM HTTP Server the file is httpd.conf.
- The installation program will take some time to install files into your system.
- After it finishes, in the last dialog box, you can choose to read the readme.txt file or click **Finish** to complete the installation.

At that point, the Websphere Application server is ready. Go to the next section for a first look at the WebSphere Application server.

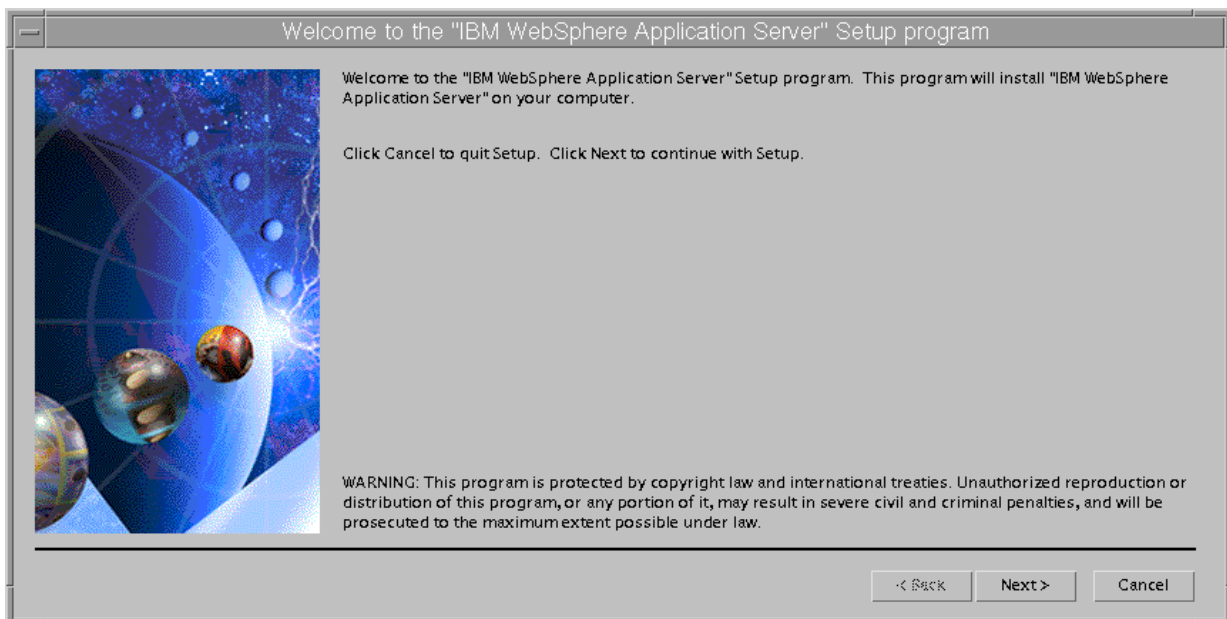


Figure 51. WebSphere Application Server Installation on AIX - The Welcome Page

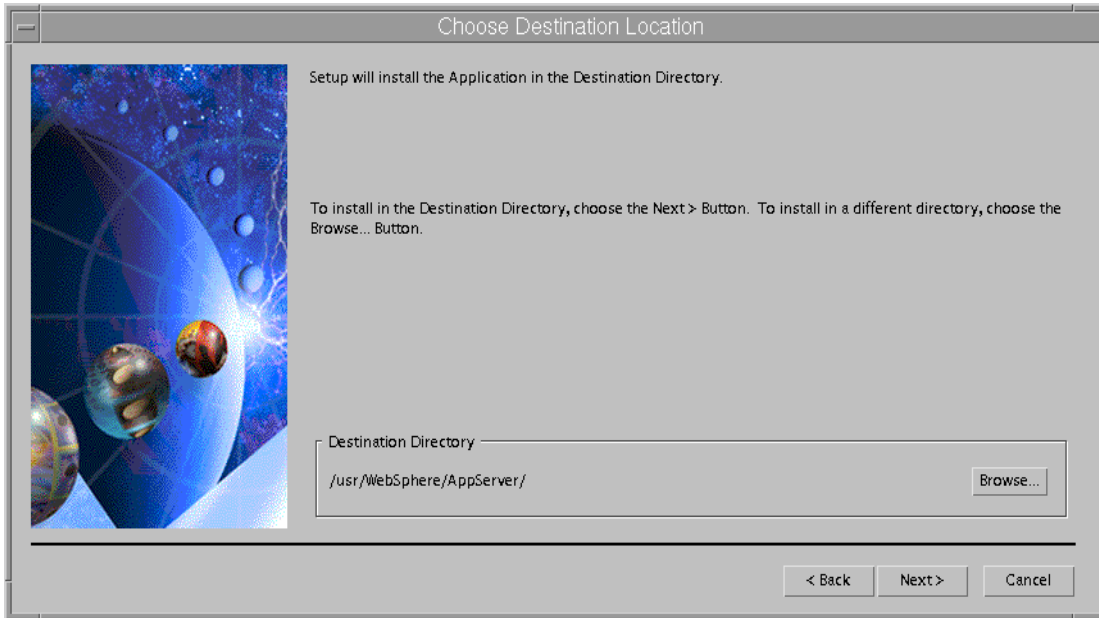


Figure 52. WebSphere Installation on AIX - Specifying WebSphere Root Directory

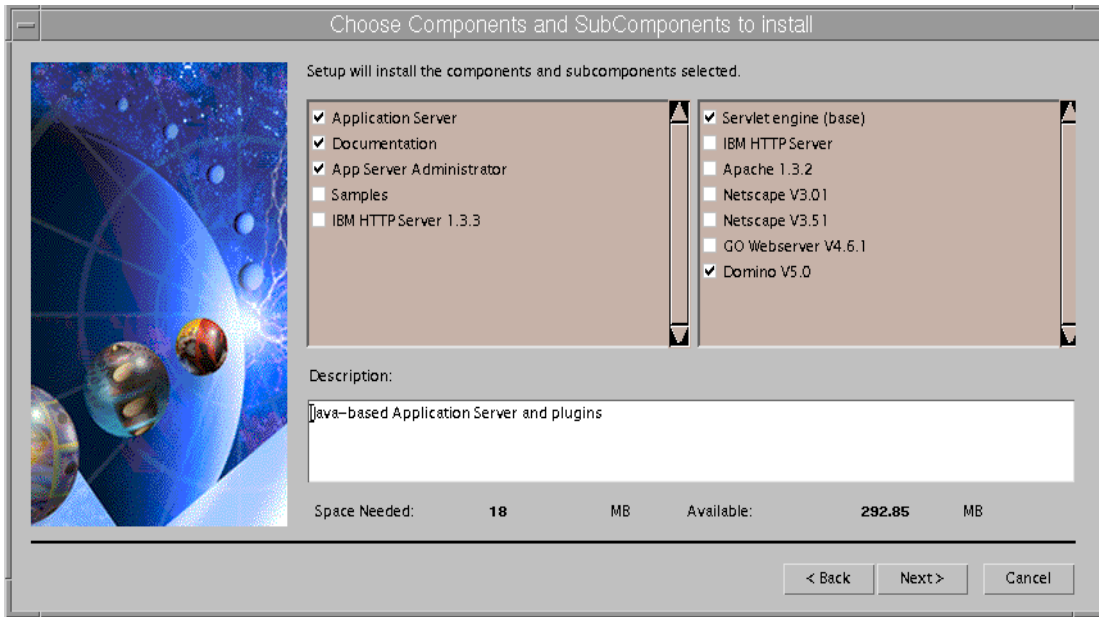


Figure 53. WebSphere Installation on AIX - Selecting Components to Install

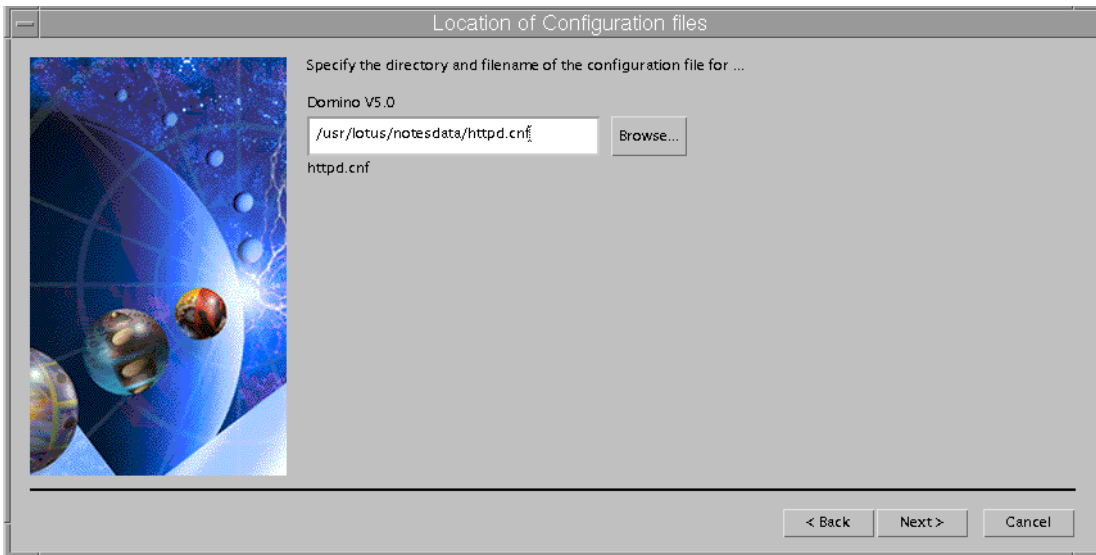


Figure 54. WebSphere Installation - Specifying Web Server Configuration File



Figure 55. WebSphere Installation on AIX - The Final Dialog Box

2.5 Using WebSphere for the First Time

After installing all of the infrastructure and WebSphere, it is time to take your first tour of WebSphere.

You should try to verify that your installation worked. Open a Web browser and go to the location: `http://<hostname>/servlet/SimpleServlet`. If the system is working, it should show a simple message "This is output from SimpleServlet". More complex samples can be found in `http://<hostname>/IBMWebAS/samples/`. For the time being, try to look at the WebSphere Administration tool.

The WebSphere Application Server Administration tool is a Web application for configuring and monitoring the Application Server. To call this application, use

your browser and go to `http://<hostname>:9527`. This will bring up the WebSphere Administration tool login page (Figure 56 on page 83):

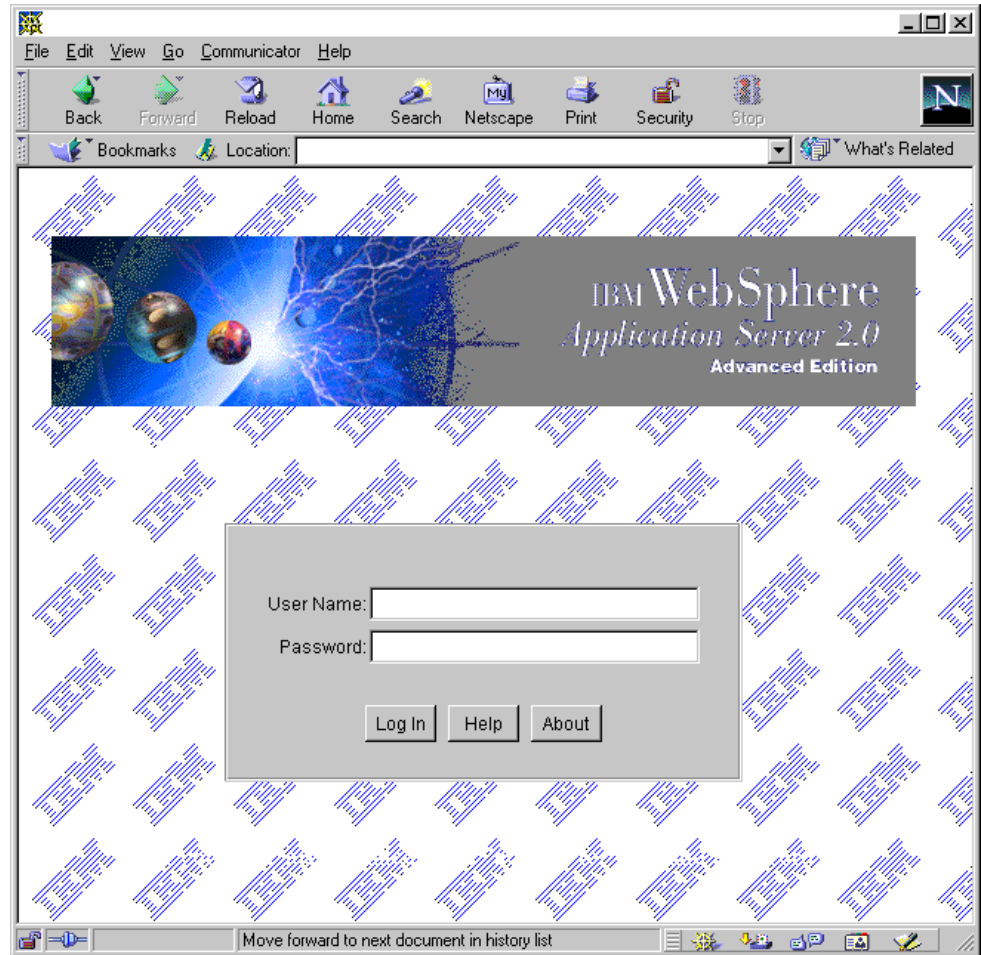


Figure 56. WebSphere Administration Tool Login Page

By default, the first time the user ID is *admin* and the password is also *admin*. Enter the Administration tool by using that user ID and password. The Administration tool (Figure 57 on page 84) consists of two frames. The left frame is for navigation. The right frame is for configuration forms.

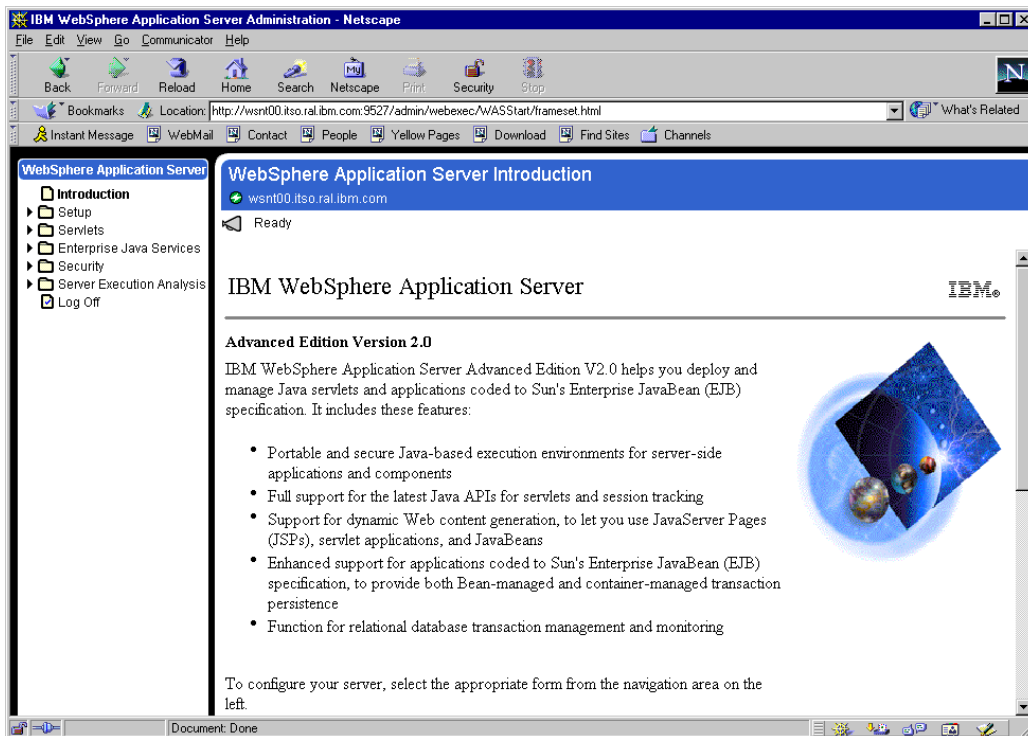


Figure 57. WebSphere Administration Tool

You can now configure WebSphere for the first time. You can set up the WebSphere Java environment, which is an important configuration in WebSphere. We also set up the Java environment in later chapters.

- On the left frame, click **Setup** to list Setup menus, then select **Java Engine** to invoke the Java Engine configuration form (Figure 58 on page 85).
- On the Path tab, there are two important parameters: Application Server Class Path and User Libpath.
- Append any additional Java library that your applications need into Application Server Class Path. For example, add D:\SQLLIB\java\db2java.zip to the end of Application Server Class Path.
- Append related native implementation library paths into User Libpath. For example, use D:\SQLLIB\bin to inform the WebSphere the location of db2jdbc.dll which is the native API for DB2 JDBC driver.
- After that, click **Save** and **Log Off** from the Administration tool.

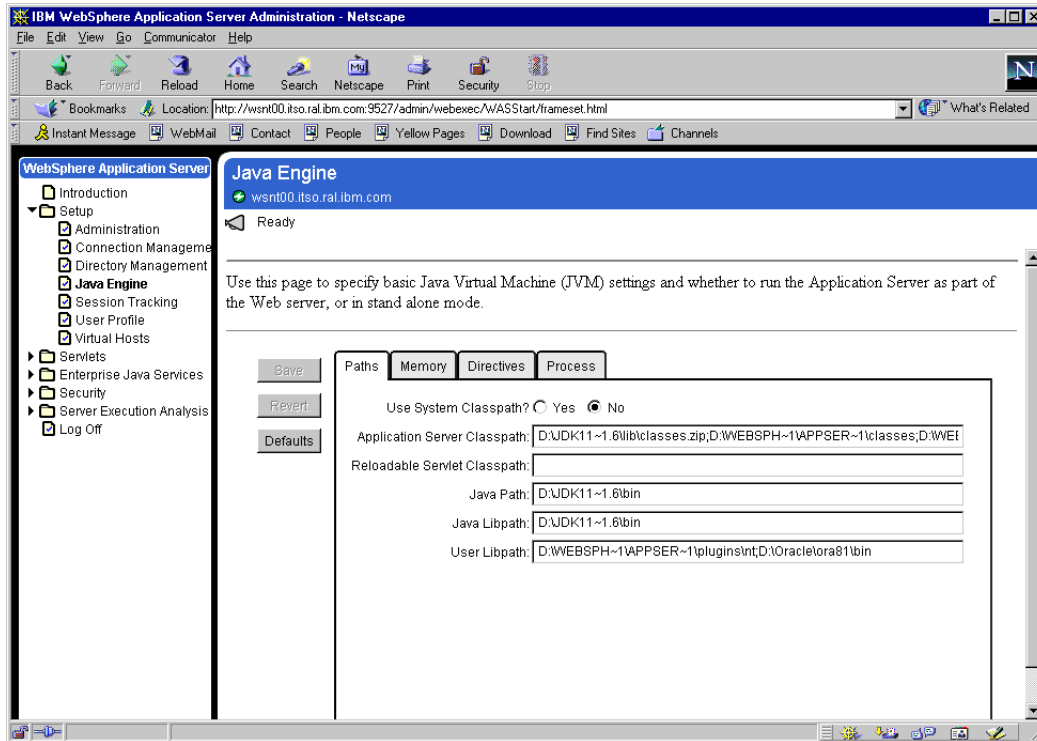


Figure 58. Setting Up Java Engine Configuration in WebSphere Administration Tool

Now explore several WebSphere directories that may be important when you are deploying applications:

- <ASROOT>/classes is the root directory for your application classes.
- <ASROOT>/servlets is the root directory for servlets. You should put servlets under this directory.
- <ASROOT>/lib contains WebSphere Java libraries. If you are not using VisualAge for Java, you might need these libraries for developing Java programs on WebSphere.
- <ASROOT>/plugins contains several plug-in modules such as plug-ins for Web servers.
- <ASROOT>/web contains WebSphere's own Web documents, help pages and utility class libraries. It is not intended for storing your Web documents.

There are several property files under <ASROOT>/properties directory. In fact all parameters configured using the WebSphere Administration tool are stored in these property files. Some important property files are:

- <ASROOT>/properties/bootstrap.properties contains basic WebSphere parameters.
- <ASROOT>/properties/ejs/ejs.properties contains EJB configurations.
- <ASROOT>/properties/server/servlet/servletservice/servlets.properties contains servlet setup.

At this point, you are ready to use WebSphere Application Server.

2.6 Setting Up a Development System

WebSphere allows you to run servlets and Enterprise Java Bean (EJB) applications, as well as general Web applications. In this environment, Java is the underlying programming language. There are many Java development tools available on the market. To develop applications in WebSphere, the tool should support servlets, JSPs, and EJB application developments. If your application requires enterprise access to various platforms and distributed components technology, the tool should also have various connectors and CORBA support.

In general, we can divide Web application development into two main fields: presentation and content generation. In developing presentation components, you work with HTML, page layout, graphics, servlets, Java beans and multimedia. In content generation, you deal mainly with data and transactions.

IBM comes with two solutions: a complete Java development tool, VisualAge for Java for working with data processing and transaction logics, and WebSphere Studio for working with presentation components.

VisualAge (VA) for Java enables you to create Web-enabled enterprise applications with proven support for building Java components. It is available on many platforms: OS/2, Windows NT, AIX, and OS/390. VA for Java is the best choice for building applications that require strong integration with other IBM products. For more information about VA for Java, you can visit

<http://www.software.ibm.com/ad/vajava>.

VA for Java 2.0 is now available. It includes:

- Visual servlet builder
- Tools for creating, testing and deploying EJB
- WebSphere Test Environment
- Enterprise Access Builder for various connectors

The WebSphere Studio provides tools to develop presentation components. The WebSphere Studio 3.0 includes:

- Visual page designer that supports dynamic HTML and Java Server Pages (JSP). You can visually design page layouts using HTML and JSP.
- Site construction management. You can import existing site contents, update links automatically, archive a site into a single file and publish the site information.
- Enhanced team development features that provide a common view of work across the entire team and source control.

In this section, we describe how to set up a development environment for building applications on top of WebSphere Application Server. We don't discuss Java programming techniques, and how to use the tools. For more information on that, there are several IBM redbooks on VisualAge for Java programming. In particular, you can read *Programming with VisualAge for Java Version 2*, SG24-5264 for an introduction on VA for Java; or *Using VisualAge for Java Enterprise Version 2 to Develop CORBA and EJB Applications*, SG24-5276 for developing EJB applications. You can obtain these books online at <http://www.redbooks.ibm.com>.

2.6.1 Setting Up VisualAge for Java

There are three editions of VA for Java 2.0: Entry Edition, Professional Edition and Enterprise Edition. To develop applications in the WebSphere environment, we recommend you use the Enterprise Edition of VA for Java 2.0. In addition, you need to install the Enterprise Update for VisualAge for Java that you can obtain from <http://www.developer.ibm.com/java>.

2.6.1.1 Adding Features into VA for Java

To develop applications in the WebSphere environment, you should add the following VA for Java features into your work space:

- IBM EJB Development Environment 1.1
- IBM Servlet Builder 2.1
- IBM Servlet Builder Examples 2.1
- IBM Servlet IDE Utility Class Libraries 2.0.3
- IBM IDL Development Environment 2.0
- IBM IDL Development Environment Examples 2.0
- IBM WebSphere Test Environment 1.1
- IBM Common Connector Framework 2.0
- IBM Enterprise Access Builder 2.0
- IBM Enterprise CICS Access Builder Library 1.0
- IBM Java Record Library 2.0
- CICS Connector 3.0.1
- Encina Connector 2.1.1

To add these features to your work space, perform the following steps:

- In the Workbench, select **File > Quick Start** or press **F2** to invoke the Quick Start dialog box (Figure 59 on page 88).
- Select **Features** in the left pane and **Add Features** in the right pane.
- This will bring up a dialog box containing a list of additional features (Figure 60 on page 88). Select the features that you want to add and click **OK**.

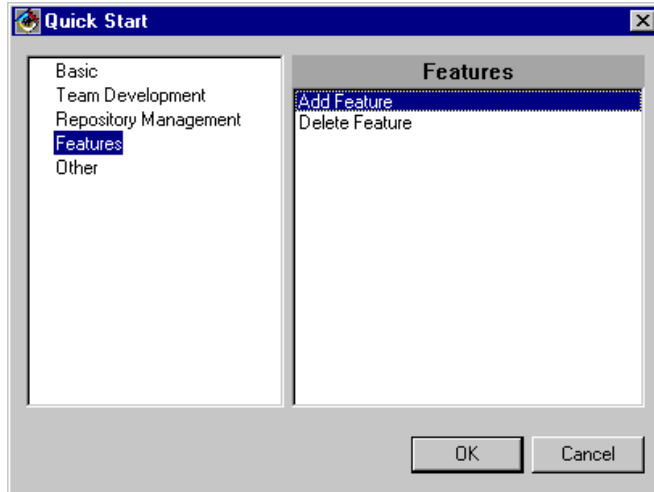


Figure 59. Quick Start Dialog Box

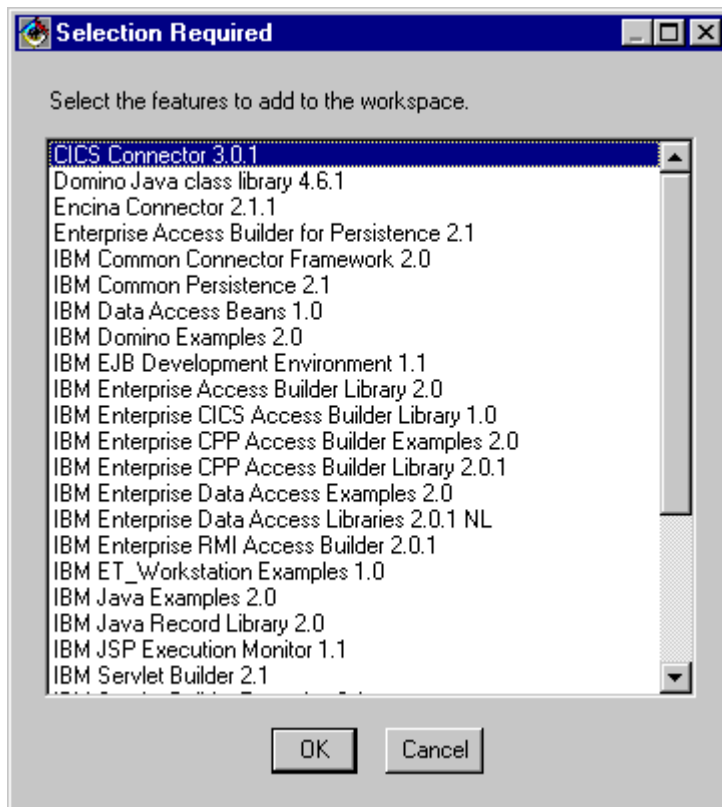


Figure 60. Adding Features into VA for Java WorkBench

After installing the features, there will be a new tab sheet for EJB in the Workbench (Figure 61 on page 89). In that sheet, you can create, test and deploy EJB applications. For instructions on using this feature, see *Using VisualAge for Java Enterprise Version 2 to Develop CORBA and EJB Applications*, SG24-5276.

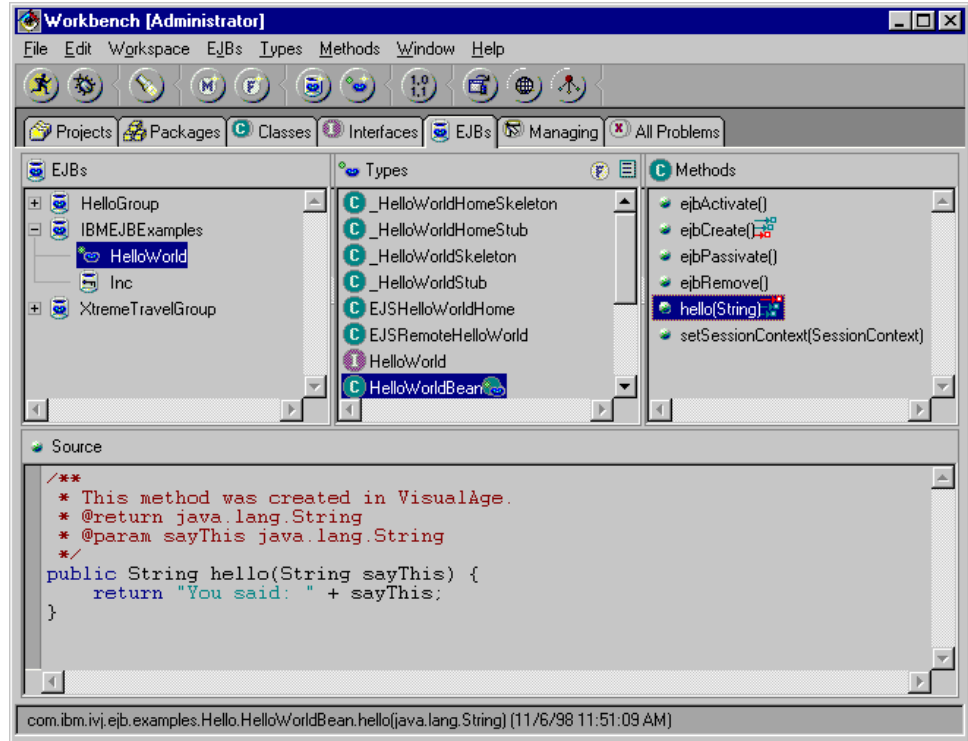


Figure 61. VA for Java Workbench

2.6.2 Setting Up and Using Command Line Session

As an alternative (though not recommended), some developers may occasionally use a command line compiler and tools. It may be useful for:

- Integrating with third-party products such as using the SQLJ precompiler (which is invoked from the command line).
- Debugging EJBs, which are developed using tools other than VA for Java, to run on WebSphere EJS.

2.6.2.1 Setting Session CLASSPATH Variables

For compiling and running applications from the command line, you should set the session CLASSPATH variables. For the Windows NT platform, create a batch file with the following contents:

```
set WASL=D:\WebSphere\AppServer\lib
set DEPL=D:\WebSphere\AppServer\deployedEJBs
set CLASSPATH=%CLASSPATH%;%WASL%\ibmwebas.jar;%WASL%\ejb.jar
set CLASSPATH=%CLASSPATH%;%WASL%\jst.jar;%WASL%\jsdk.jar;%WASL%\xml4j.jar
set CLASSPATH=%CLASSPATH%;%WASL%\databeans.jar
REM
REM Put other Java libraries here ... e.g. for databases
REM
set CLASSPATH=%CLASSPATH%;D:\sqllib\java\db2java.zip
REM
REM Put Your deployed EJB JAR files below ...
REM
set CLASSPATH=%CLASSPATH%;%DEPL%\HelloServer.jar
set CLASSPATH=%CLASSPATH%;%DEPL%\<your_deployed_ejb_jar>
...
```

For the AIX platform, create a batch file with the following contents:

```
export WASL=/usr/WebSphere/AppServer/lib
export DEPL=/usr/WebSphere/AppServer/deployedEJBs
export CLASSPATH=$CLASSPATH:$WASL/ibmwebas.jar:$WASL/ejs.jar
export CLASSPATH=$CLASSPATH:$WASL/jst.jar:$WASL/jsdk.jar:$WASL/xml4j.jar
export CLASSPATH=$CLASSPATH:$WASL/databeans.jar
#
# Put other additional Java libraries here ... e.g for databases
#
set CLASSPATH=$CLASSPATH;/home/db2inst1/sqlllib/db2java.zip
#
# Put Your deployed EJB JAR files below ...
#
export CLASSPATH=$CLASSPATH;$DEPL/HelloServer.jar
export CLASSPATH=$CLASSPATH;$DEPL/<your_deployed_ejb_jar>
...

```

Starting Enterprise Java Service (EJS) Manually

The EJS is WebSphere EJB Server. You can start EJS manually from the command line. Before that, make sure that WebSphere is not running. To start the EJS, perform the following steps:

- Change the directory to the WebSphere root directory.
- Set session CLASSPATH variables as listed above.
- Start the EJS Location Service Daemon:

```
java com.ibm.lsd.LocationServiceDaemon -ORBListenerPort <lsd_port>
```

where <lsd_port> is the Location Service Daemon listening port. For example:

```
java com.ibm.lsd.LocationServiceDaemon -ORBListenerPort 9029
```

- Start the EJS Persistent Name Server:

```
java com.ibm.CosNaming.PersistentNameServer -ORBBootstrapPort
<bootstrap_port> -ORBPersIORHostName <lsd_host_name> -ORBPersIORPort
<lsd_port> -InitialRoot <name_server_dir>
```

where <bootstrap_port> is the port on which to start the name service server;
<lsd_host_name> is Location Service Daemon server host name;
<name_server_dir> is a directory to contain Name Server data. For example:

```
java com.ibm.CosNaming.PersistentNameServer -ORBBootstrapPort 9019
-ORBPersIORHostName wsnt00.itso.ral.ibm.com -ORBPersIORPort 9029
-InitialRoot d:\EJSNameSpace
```

- Start the EJS Server

```
java -nojit com.ibm.ejs.server.EJServer -ORBPersIORPort <lsd_port>
-ORBBootstrapPort <bootstrap_port> -file <ejs_properties_file>
```

where <ejs_properties_file> is the fully qualified file name for WebSphere EJS configuration file ejb.properties. For example:

```
java -nojit com.ibm.ejs.server.EJServer -ORBPersIORPort 9029
-ORBBootstrapPort 9019 -file
d:\WebSphere\AppServer\properties\ejb\ejb.properties
```

Chapter 3. Content Presentation

The Web technologies that are available to present information to the client are evolving at a rapid pace. First, there was HTML, which displayed only static content. Shortly after that came Common Gateway Interface (CGI), a simple protocol that can be used to communicate between Web forms and your applications. It enabled dynamic content, with some programming languages such as C and Perl. But CGI had trouble with its performance and scalability because it started and ended processes for each request. This adversely affected performance.

To solve this problem, other technologies were attempted. One of them used the Web server's APIs. That enabled the programs to run quicker, but it took a lot of effort to develop the applications and they were unique to each vendor's Web server.

Another technology was Fast CGI. It is a high-performance extension of CGI, using long-lived processes that are API-independent. But it was also hard to develop because it required specific coding or configuration on each httpd server or in each developer's environment.

The servlet is a recent Java-based technology and it runs on the servlet engine. It provides high performance and has many of the same advantages as Java. It has portability and robustness, and provides security at run time. The servlet is becoming more commonly used on Web applications. You can find more information about servlets at:

<http://java.sun.com/products/servlet/index.html>

In addition to discussing servlets in this chapter, we explain and provide examples of JSP coding and XML. This chapter shows how to use these new technologies on WebSphere.

3.1 How to Deploy and Configure a Servlet

This section shows you the steps required to deploy servlets and configure servlets in WebSphere Application Server V2.02. Most of these procedures assume that you are logged on to the WebSphere Application Server Administration interface using your administrator user ID and password. This can be achieved by going to the Web address `http://<hostname>:9527/` and typing your user ID and password in the fields provided.

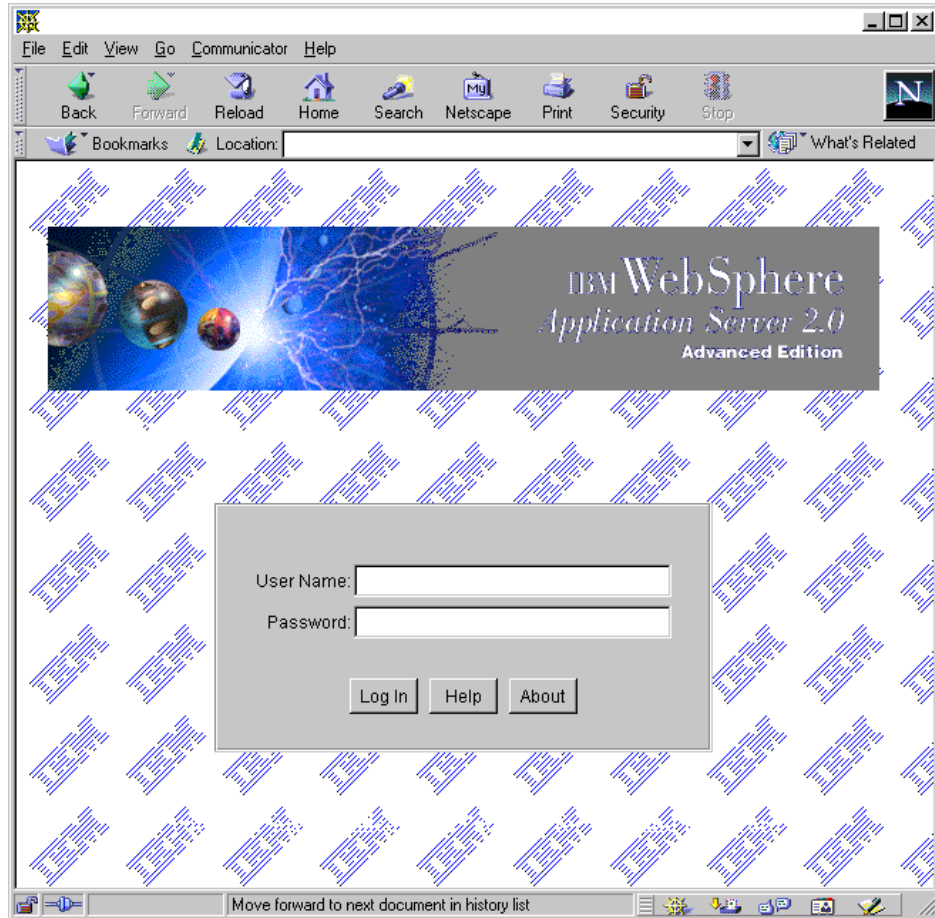


Figure 62. IBM WebSphere Application Server Administration Log In

Most of the time, the servlet is invoked from a link in HTML files (also JSP and SHTML files). We show how to deploy servlets, including related HTML files in this section.

To deploy a servlet on your application server perform the following steps:

1. Copy the class files to the application server.
2. Copy the related HTML, JSP, and SHTML files to the application server.
3. Use the Application Server Manager to configure initialization parameters and set other options. This last step is optional.

3.1.1 Placing Class Files on the Application Server

By default, the application server looks for servlet class files in the servlet root directory, `<ASRoot>\servlets`. The second step required to deploy the servlets requires you to perform the following tasks:

- Put a copy of your compiled servlet class files in the servlet root directory.
- If the servlets are in a package, mirror the package structure as subdirectories under the `servlet\` or `reloadable servlet` directory. For example, if the servlets `SendMessage.class` and `GetMessage.class` are in a package named `com.ibm.servlet.servlets.personalization.util`, copy the servlets into the

directory <ASRoot>\servlets\com\ibm\servlet\servlets\personalization\util as shown in Figure 63:

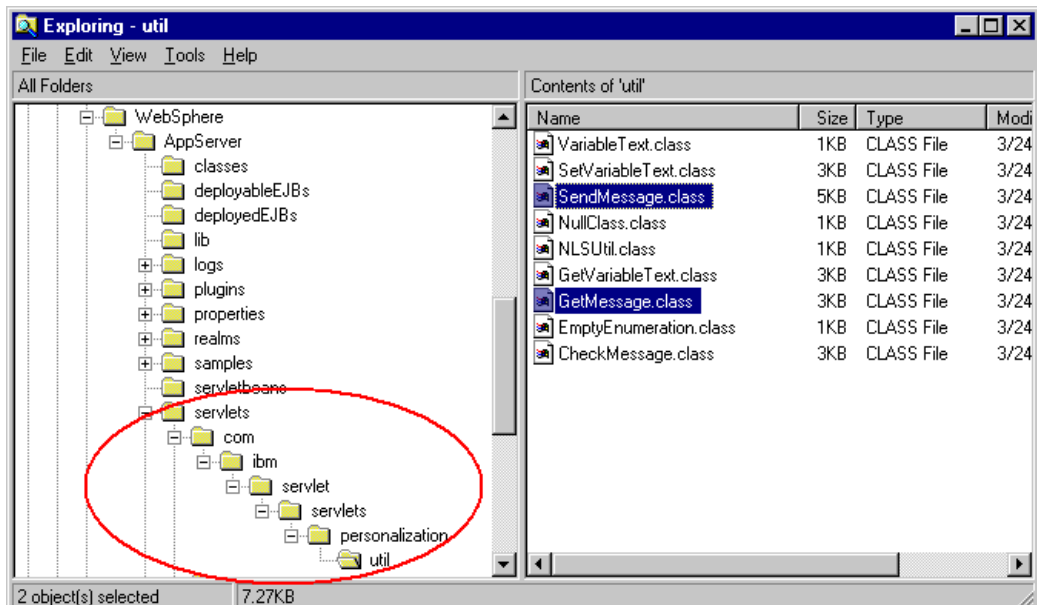


Figure 63. The Servlets in a Package

- If your servlets import non-servlet classes that you developed, it is recommended that you copy those classes to *applicationserver_root\servlets*.
- Standard out logging (Stdout) for all servlets goes to <ASRoot>\jvm_stdout.logs or to the Java console window, depending on the settings in the bootstrap.properties file. See 8.2, “The Application Server Debug Console” on page 418 for instructions on enabling Java standard out logging.

For example, if you call the *StdTest* servlet shown in Figure 64 on page 94, outputs for System.out.println are written to jvm_stdout.log and System.err.println is written to the jvm_stderr.log.

```

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

public class StdTest extends HttpServlet {
public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException,
IOException
{
    PrintWriter out;
    res.setContentType("text/html");
    out = res.getWriter();

    out.println("<html>");
    out.println("<head><title>StdTest</title></head>");
    out.println("<body>");
    out.println("<h1>StdTest</h1>");
    out.println("See jvm_stdout.log & jvm_stderr.log<BR>");
    out.println("</body></html>");
    System.out.println("This is standard output.");
    System.err.println("This is standard err.");
}
}

```

Figure 64. StdTest.java

```

null
Error finding default business: bean method raised unchecked exception;
nested exception is:
:
:
read properties file: can't find resource for IBMConnMgrTestStrings_en_US
This is standard output.

```

Figure 65. jvm_stdout.log

```

:
:
at com.ibm.servlet.engine.api.ServerEntry.service(Compiled Code)
at com.ibm.servlet.engine.nativeEntry.NativeServerEntry.service(Compiled Code)
at com.ibm.servlet.engine.outofproc.OutOfProcThread.run(Compiled Code)
This is standard err.

```

Figure 66. jvm_stderr.log

3.1.2 Placing HTML, JSP, and SHMTL Files on the Application Server

The next step in the process is to copy the HTML, JSP, and SHMTL files for the servlet to the Web server's HTML document root directory, <DocRoot>. This directory is determined by your specific server configuration (the settings for path, alias, and virtual hosting rules).

You should code your servlets to use the method that returns the location of the servlet's HTML file. For example:

```

    ServletRequest.getRealPath("/my.html")

```

where "/my.html" is the name of the servlet's HTML file.

The Web server document root directory is accessible from your Web browser by opening the URL:

`http://<hostname>/`

You can copy HTML files into subdirectories relative to the document root directory in:

`<DocRoot>/morefiles`

To open the HTML file from a browser, specify the relative directory in the URL. For example, if you added the morefiles subdirectory under the document root directory and placed an HTML file there (as shown in Figure 67), open the following URL to view the file (Figure 68):

`http://<hostname>/morefiles/myhtml.html`

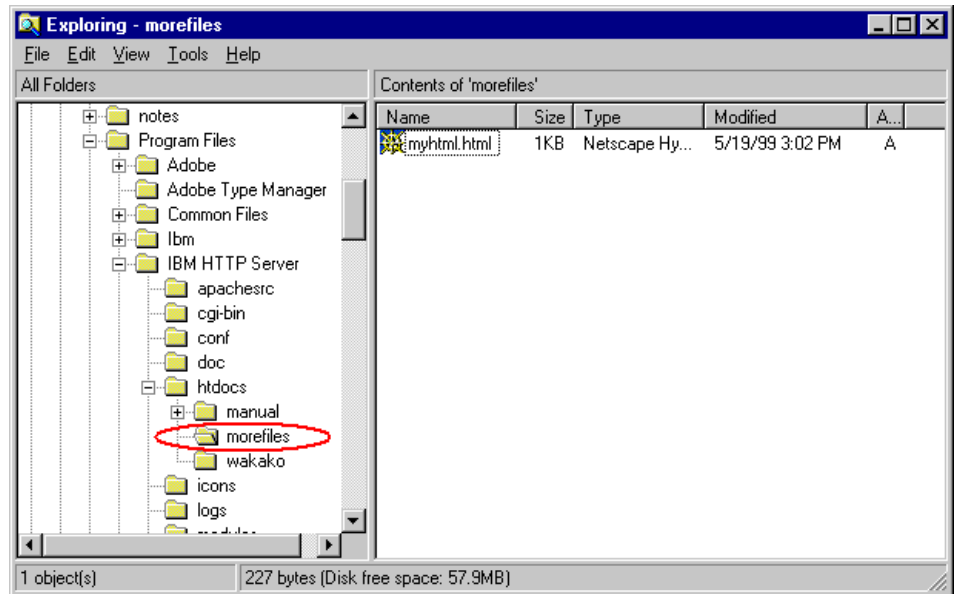


Figure 67. `C:\Program Files\IBM HTTP Server\morefiles\myhtml.html`

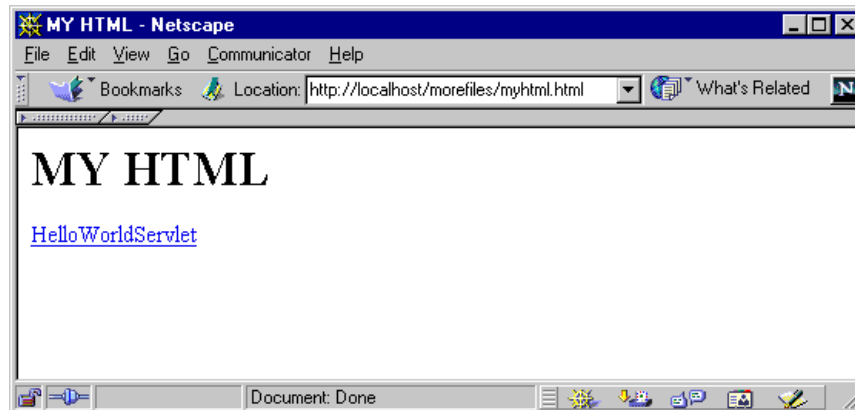


Figure 68. `my.html` Called by `http://<hostname>/morefiles/myhtml.html`

3.1.3 Configuring a Servlet

The fourth step is to configure the servlets. If you want to load a servlet from a JAR or SER file on a remote system (3.1.3.8, “Loading Remote Servlet” on page 103) or set servlet initialization parameters (see 5.3.1.3, “Creating Multiple Site Queues in a Server” on page 288), use the Application Server Manager to configure the servlet. You can also use it to implement the XML servlet configuration.

Note: If your application server is running on Netscape Enterprise or Netscape FastTrack Server under Sun Solaris and you specify a JAR file for the remote load, be sure that the JAR file was created with the no compression flag (-0) set. If compression was specified when the JAR file was created, the browser will return an error 500 when you try to invoke the servlet.

3.1.3.1 Configuring Servlets

To see a servlet's settings, choose the servlet from the Servlet Names list on the left side of the Configuration page. The servlet details display on the right side of the page. You can add servlets to the list and modify information for servlets that are in the list.

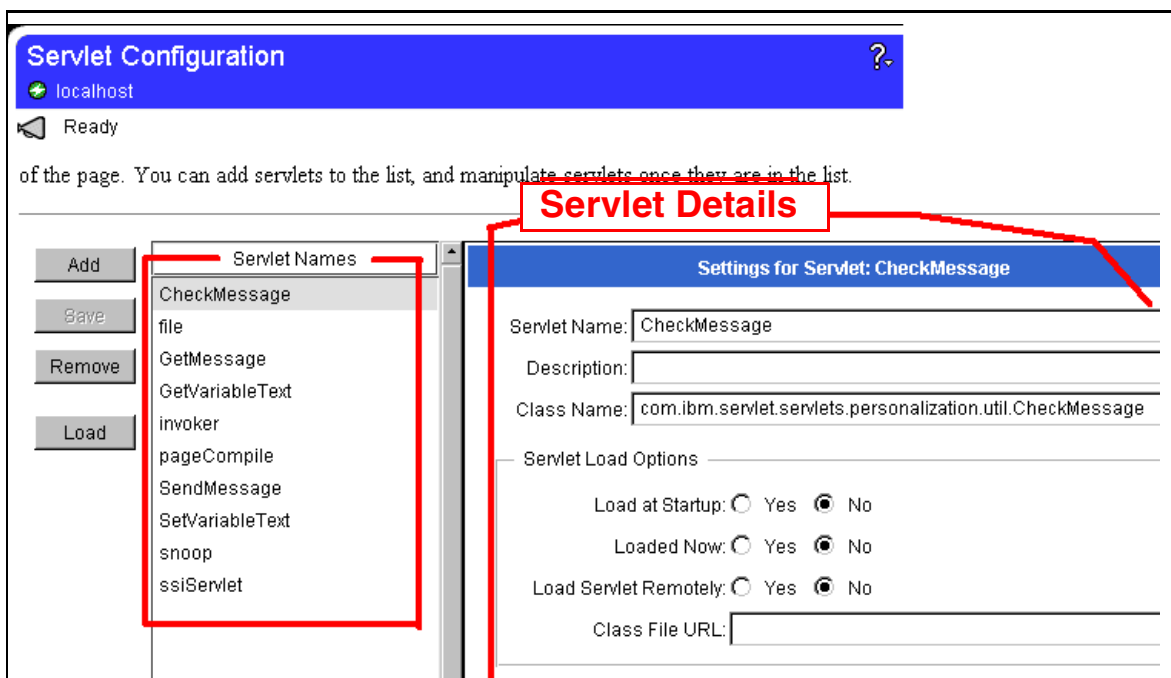


Figure 69. Configuring Servlets

You can:

- Add servlets.
- Delete servlets.
- Modify servlets.
- Load servlets.
- Unload servlets.

3.1.3.2 Adding Servlets

1. Click the **Servlets** -> **Configuration** page as shown in Figure 70 on page 97.
2. Click the **Add** button to display the Add a New Servlet dialog box as shown in Figure 71 on page 98.

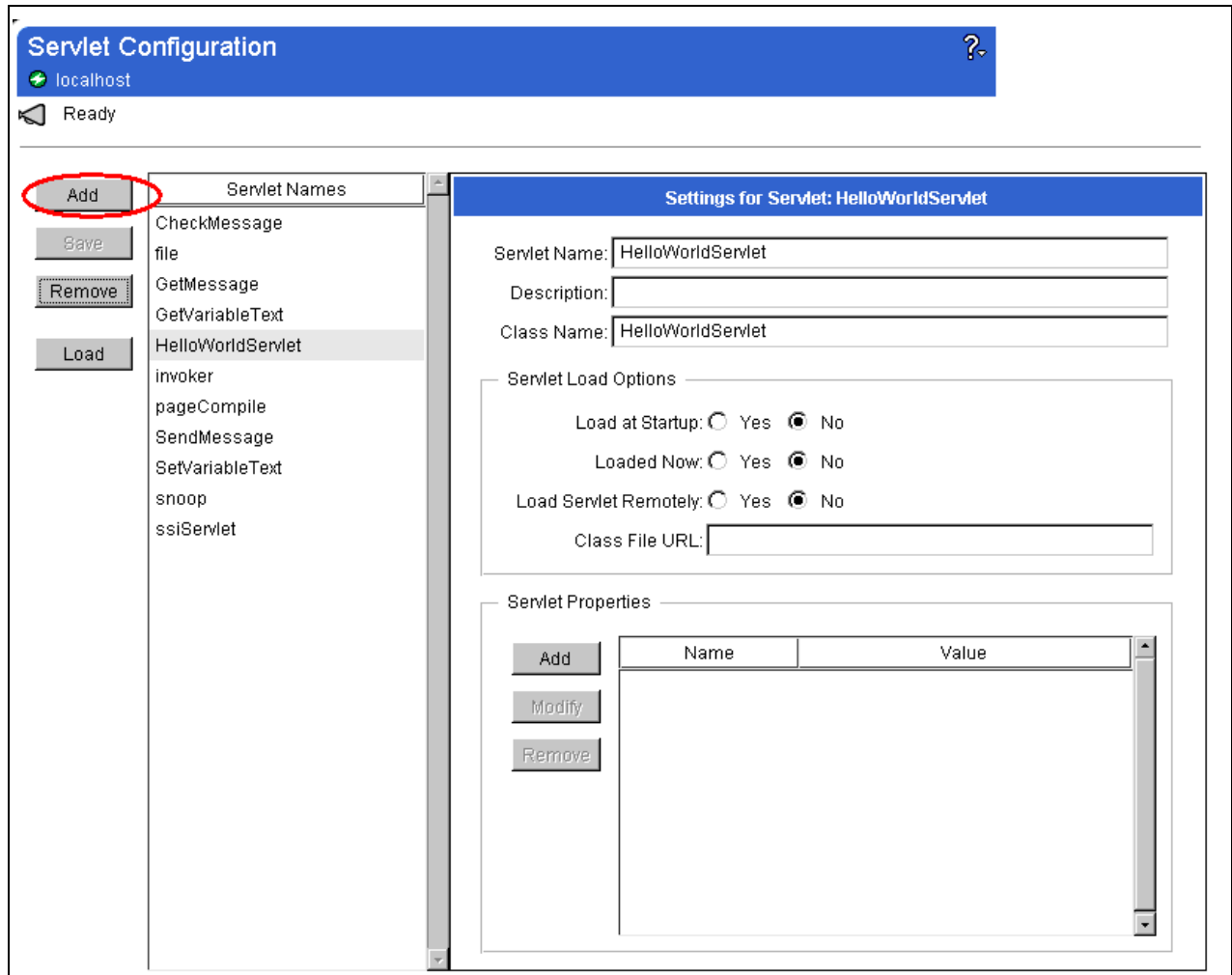


Figure 70. Add Button to Display the Add a New Servlet Dialog Box

3. In the Servlet Name field, enter the unique name of the servlet to add.

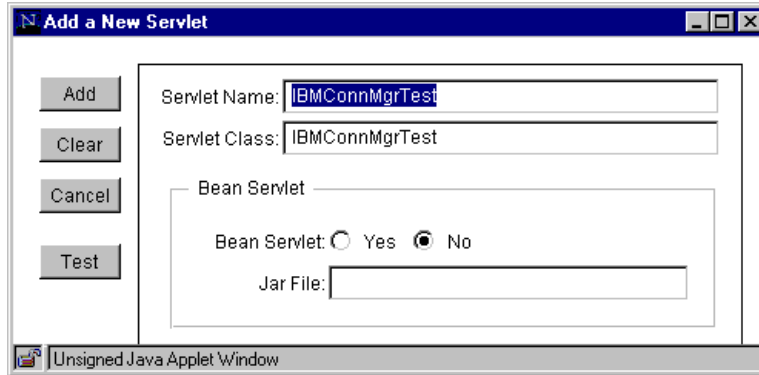


Figure 71. Add a New Servlet

4. In the Servlet Class field, enter the name of the Java class for the servlet. This name consists of the package name without the `.class` extension. For example, `sun.server.http.FileServlet` is a valid class name for `sun.server.http.FileServlet.class`.
5. Specify whether the servlet is a JavaBean. If it is, specify the full path to the JAR file containing the bean. You can see this example in 3.1.3.3, “Adding Bean Servlet” on page 99.
6. Optionally, use the **Test** button to confirm that the Application Server can find your servlet. If you get a message that your servlet code can’t be found, check the data you have entered.
7. Click the **Save** button.
8. You should see your servlet name appear in the Servlet Names field as shown in Figure 72 on page 98.

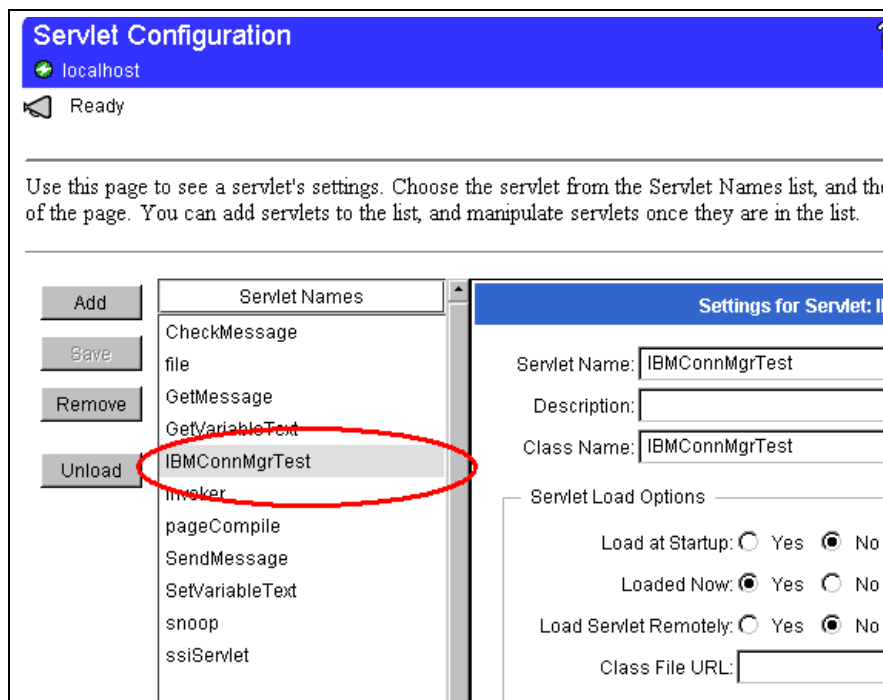


Figure 72. Added Servlet

3.1.3.3 Adding Bean Servlet

There is an interesting sample of Bean servlet in <ASRoot>/servletbean. We show how to add a Bean servlet by adding this servlet:

1. Copy SampleBean.jar to <ASRoot>/servlets from <ASRoot>/servletbean.
2. Click the **Add** button on the Servlet Configuration page. The following dialog box appears:

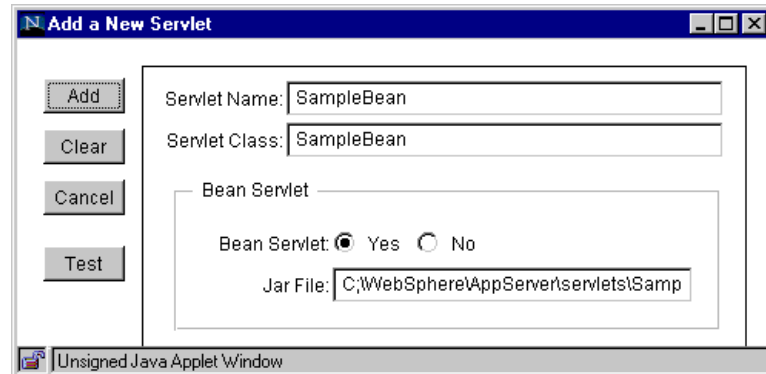


Figure 73. Add a New Servlet

3. Check **Yes** for the Bean Servlet option and specify the absolute path of the SampleBean.jar file. In this case, it is C:\WebSphere\AppServer\servlets\SampleBean.jar.
4. Click the **Test** button. If the Application Server can't find the .jar file or specified class in the .jar file, the following error message will appear:

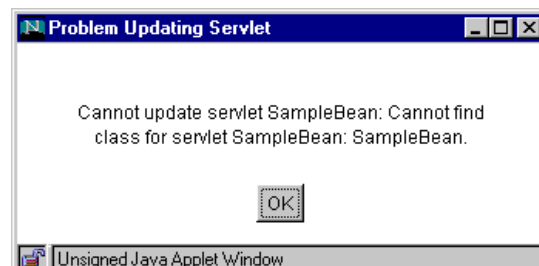


Figure 74. "Test" Failed

If the Application Server finds the class, then the following message appears:

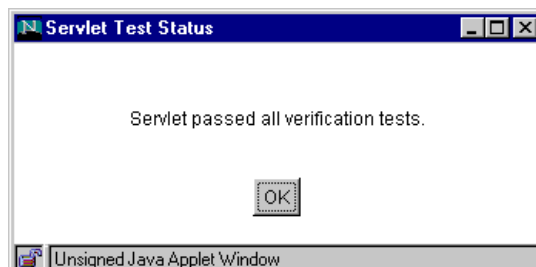


Figure 75. "Test" Succeeded

5. Click the **Add** button on the dialog box (Figure 73 on page 99). You can see the SampleBean on Servlet Configuration page (Figure 76 on page 100).

- In the Servlet Properties field, the properties for the SampleBean are shown. When the Bean is added it is introspected.

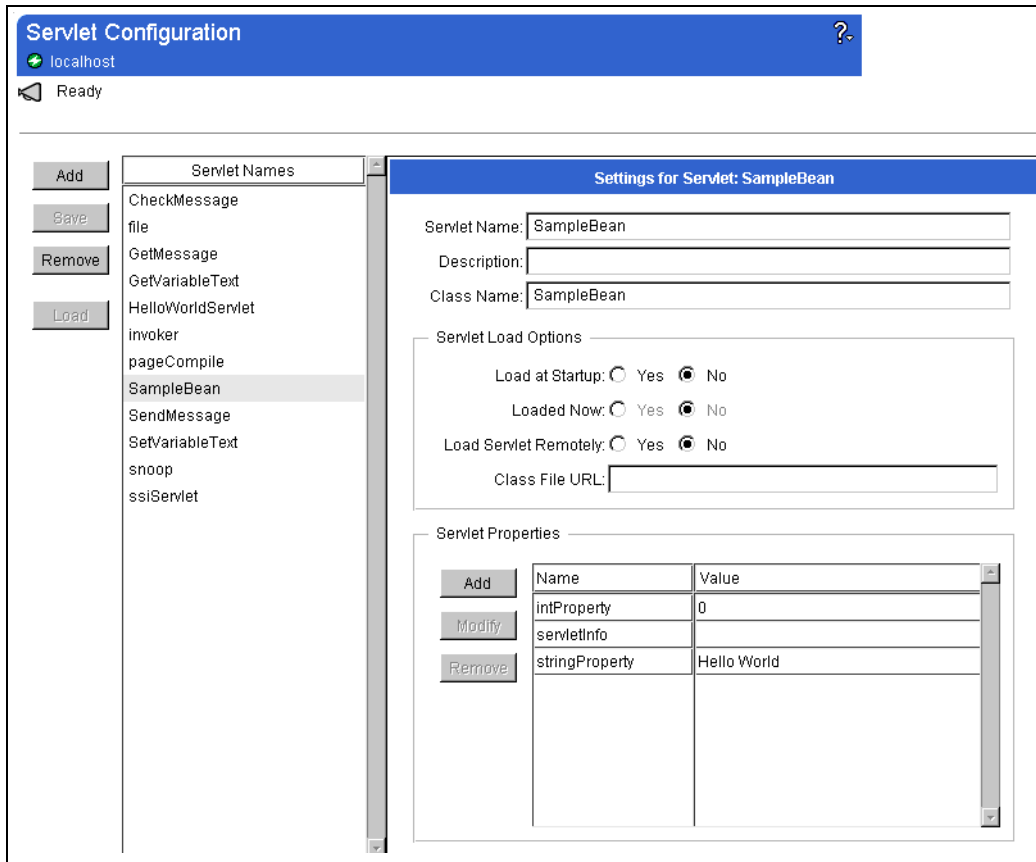


Figure 76. SampleBean Servlet Added on Configuration Page

- When you invoke this servlet from the URL `http://<hostname>/servlet/SampleBean` you should see the following window:

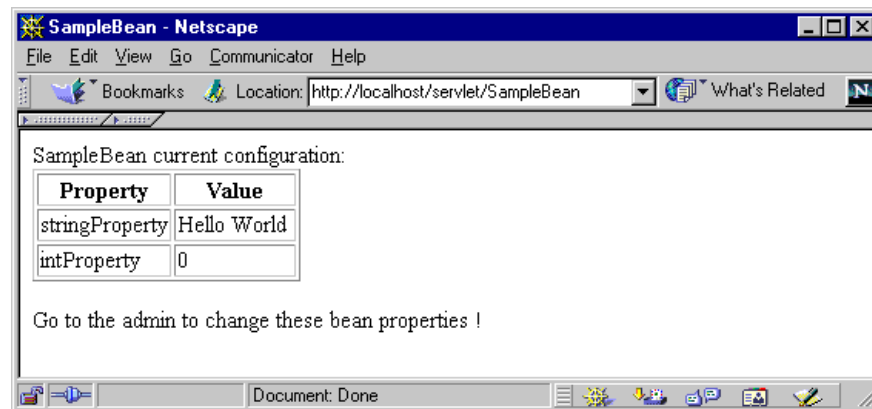


Figure 77. The Result of SampleBean Servlet

3.1.3.4 Modifying Servlets

In order to modify an existing servlet perform the following steps:

- Click the **Servlet Configuration** page as shown in Figure 69 on page 96.

2. From the Servlet Names column, select the servlet to modify.
3. Change the servlet settings.

The following window shows how to change property values for a Servlet Bean. For Servlet Beans the buttons in this field are disabled. Since Beans have their property definitions in their class files, it is logical that the field is disabled. You can modify the value of the property by double-clicking the value field as shown in Figure 78 on page 101:

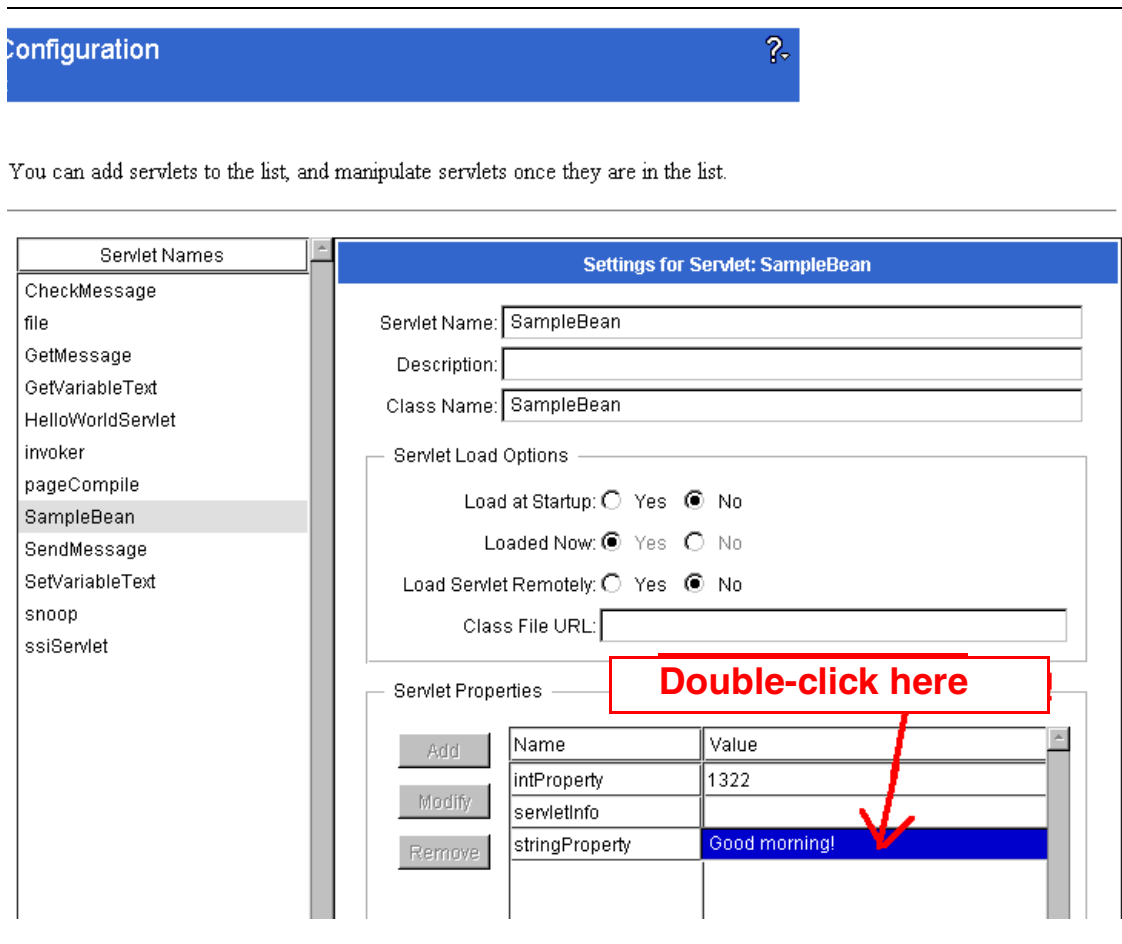


Figure 78. How to Change Servlet Properties

4. Click the **Save** button.
5. Open `http://<hostname>//servlet/SampleBean` on your browser. The following window appears:

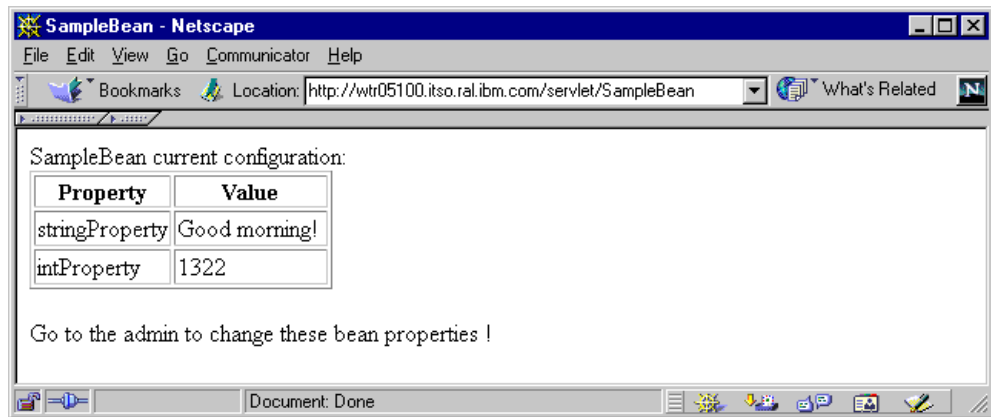


Figure 79. The Properties Can Be Changed from Servlet Configuration Page

After this modification you can see that SampleBean.ser was created in <ASRoot>/servletbeans. This ".ser" file is an instance of the SampleBean.class created with the Serialize technology of Java.

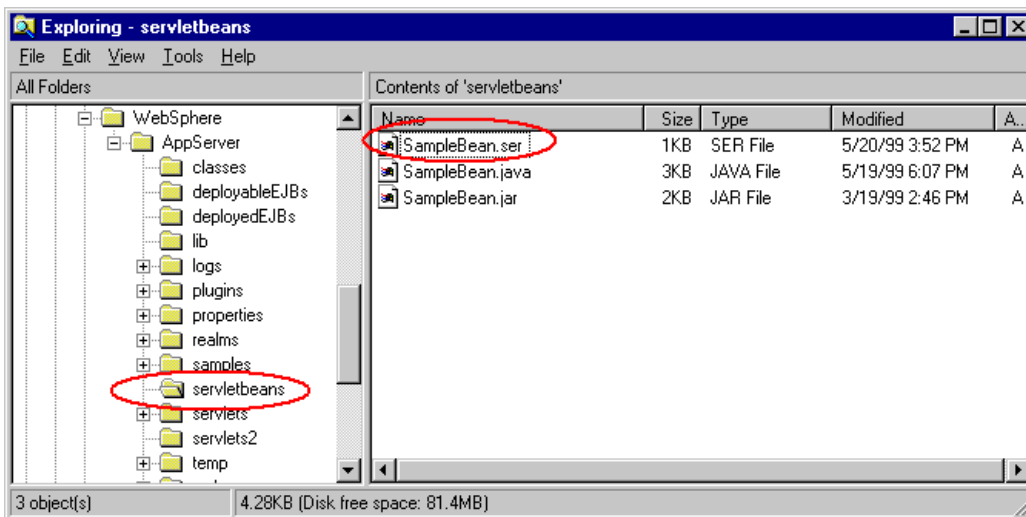


Figure 80. SampleBean.ser Was Created in <ASRoot>/servletbeans

The modifications to the Beans are reset after the Application Server goes down.

Modifying Load Options

You can set the servlet to load automatically at application server startup by checking the **Load at Startup** option shown in Figure 81 on page 103.

The following items are reasons to load a servlet at startup time:

- The servlet takes time when loading because of its heavy init procedure and you don't want to affect the response time for users.
- The servlet needs an initial parameter to be explicitly specified.
- You shouldn't load too many servlets at once. It takes a lot of resources and causes overall performance to decline.

Servlet Load Options

Load at Startup: Yes No

Loaded Now: Yes No

Load Servlet Remotely: Yes No

Class File URL:

Figure 81. Servlet Load Options

The Load Servlet Remotely option refers to the ability of the Web server to load the servlet from a remote location. The servlet must be in a JAR file for remote loading.

3.1.3.5 Deleting Servlets

To delete a servlet perform the following steps:

1. Click the **Servlet Configuration** page (shown in Figure 69 on page 96).
2. From the Servlet Names column, select the servlet to delete.
3. Click the **Remove** button.
4. Confirm the deletion by clicking **Yes**.

3.1.3.6 Loading Servlets

To load a servlet:

1. Click the **Servlet Configuration** page (shown in Figure 69 on page 96).
2. From the Servlet Names column, select the servlet to load.
3. Click the **Load** button.

3.1.3.7 Unloading Servlets

To unload a servlet:

1. Click the **Servlet Configuration** page.
2. From the Servlet Names column, select the servlet to unload.
3. Click the **Unload** button.

Note: Do not unload the invoker servlet. This is the servlet, called in every servlet request, to "invoke" a servlet. If you unload this servlet, the next servlet call takes time to invoke the invoker servlet and called servlet.

3.1.3.8 Loading Remote Servlet

You can copy the servlet class (JAR) file to a remote server and get the file when it is needed. To set up the remote loading of a servlet:

1. Create a JAR file from the servlet class file.

```
T>jar -cvf HelloWorldFromRs60004.jar HelloWorldFromRs60004.class
adding: HelloWorldFromRs60004.class (in=889) (out=521) (deflated 41%)
```

2. Copy HelloWorldFromRs60004.jar to the HTTP document directory on the remote system (not in the servlet root directory). The remote server where you placed the .jar file doesn't have to be a WebSphere server.
3. The .jar file's permissions must be set up as read for everyone.

```
rs60004:/usr/lpp/HTTPServer/share/htdocs > ls -la HelloWorldFromRs60004.jar
-rw-r--r-- 1 root system 975 May 27 10:32 HelloWorldFromRs60004.jar
rs60004:/usr/lpp/HTTPServer/share/htdocs >
```

4. Add a new class.

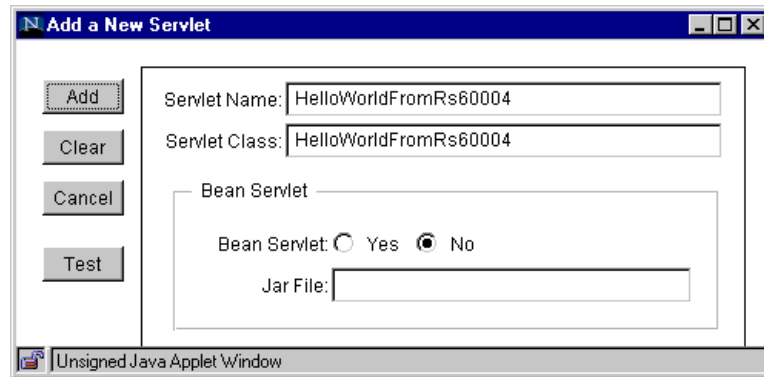


Figure 82. Add a New Servlet

5. On the Servlet Load Options check the **Yes** option for the Load Servlet Remotely field. In addition, specify the URL of the JAR file. In this example, it is <http://rs60004.itso.ra1.ibm.com/HelloWorldFromRs60004.jar>.

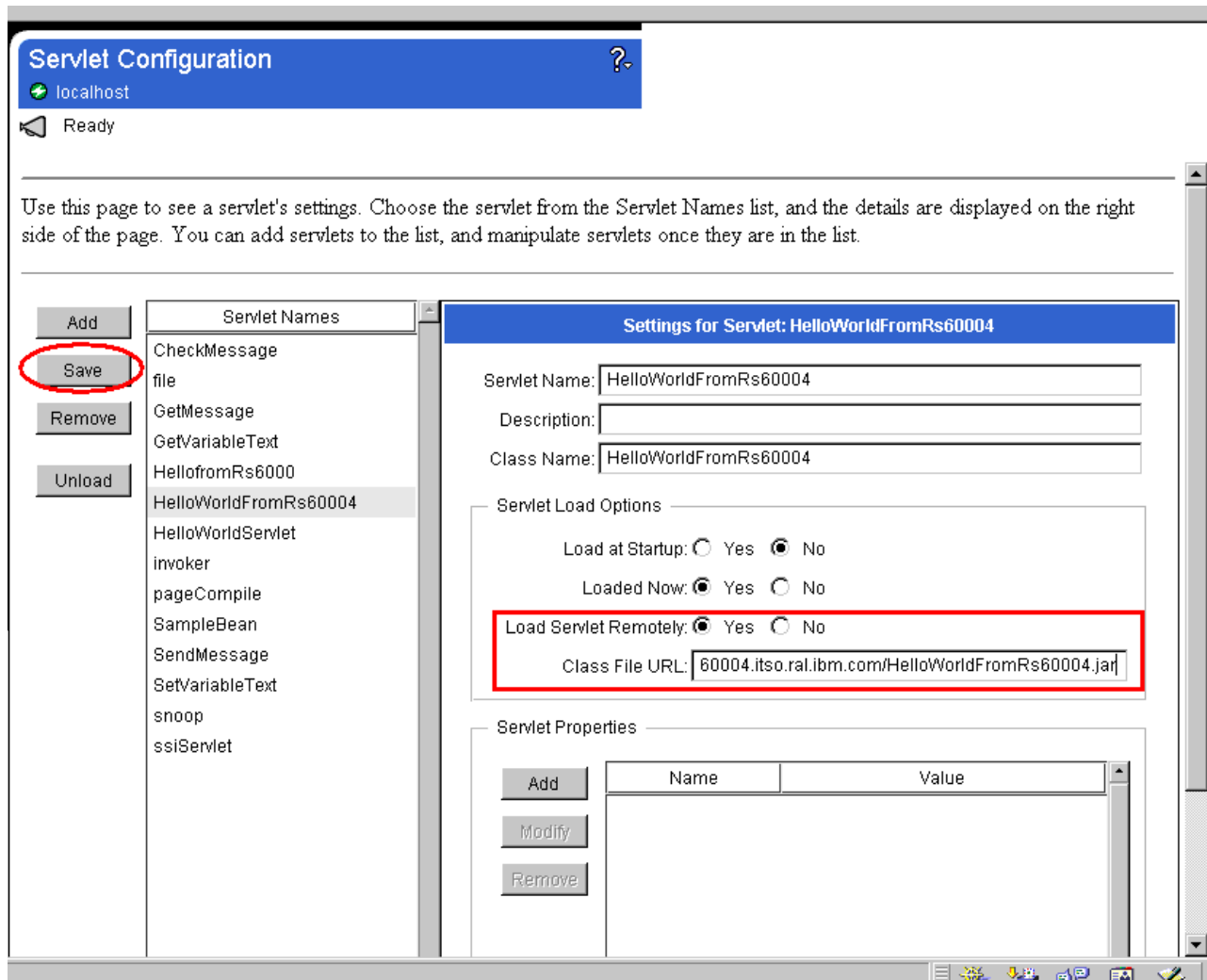


Figure 83. Specify the Class File URL and Click the Save Button

6. Click the **Save** button and go to:
`http://localhost/servlet/HelloWorldFromRs60004`
7. The following window will appear:

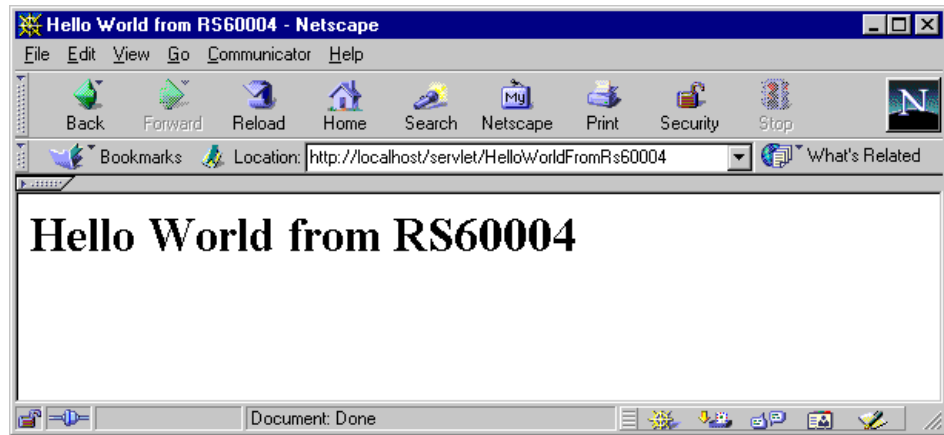


Figure 84. Remotely Load Servlet

3.1.3.9 Configuring Alternate Servlet Directories

You can put the servlet in other directories besides the <ASRoot>/servlets directory. If you do that, you need to do some more customization.

To load servlets from an alternate servlet directory, you need to configure a reloadable servlet directory. It is on the Paths tab of the Java Engine page (see Figure 85 on page 107). You get there by clicking **Setup -> Java Engine**. The servlet root directory is reloadable by default.

Specify the reloadable servlet classpath and click the **Save** button.

C:\WebSphere\AppServer\servlets2 is specified as the alternate directory in Figure 85:

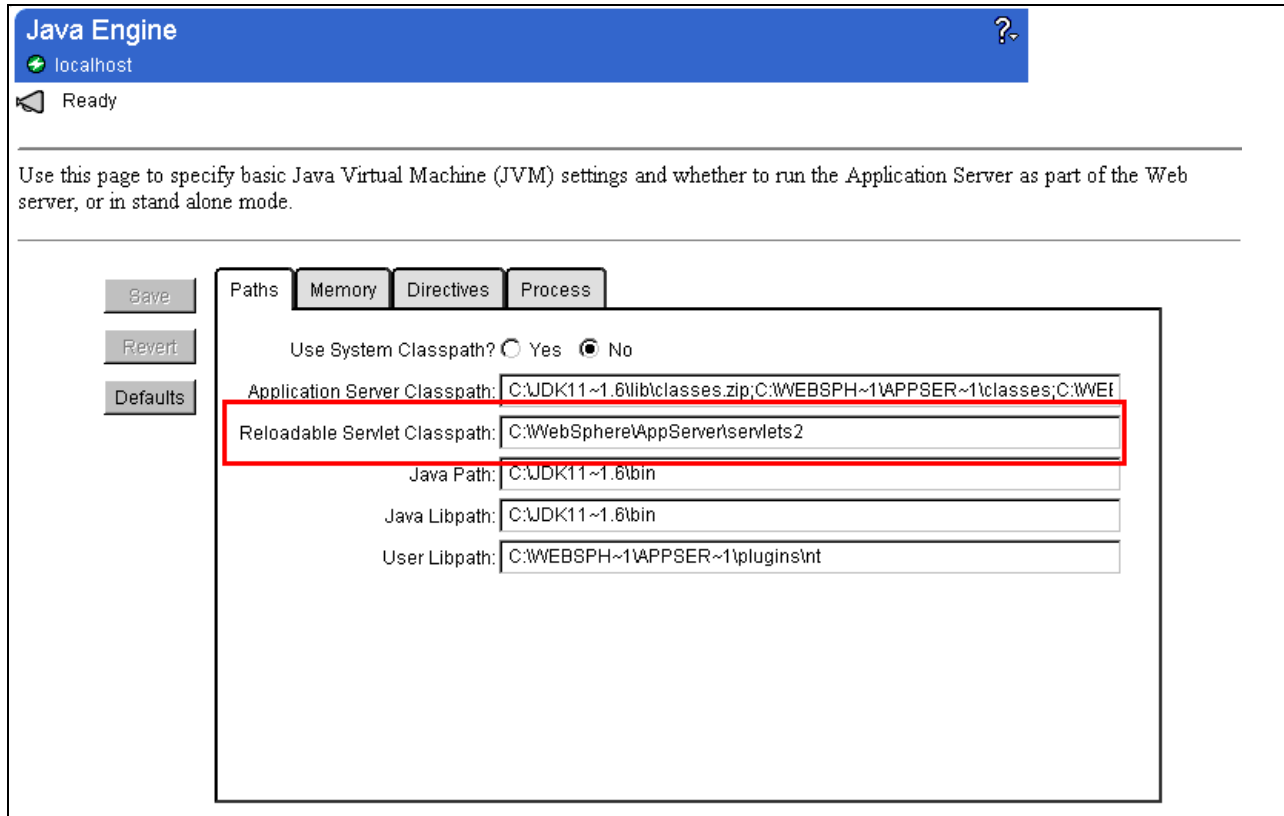


Figure 85. Reloadable Servlet Classpath

After setting it you can invoke the servlet placed in the specified directory the same way as you would invoke the servlet placed in the <ASRoot>/servlet directory. For example, if you copy the HelloWorldServlet2.class to the C:\WebSphere\AppServer\servlets2 directory, the HelloWorldServlet2 servlet can be invoked by

`http://<hostname>/servlet/HelloServlet2`

See Figure 87.

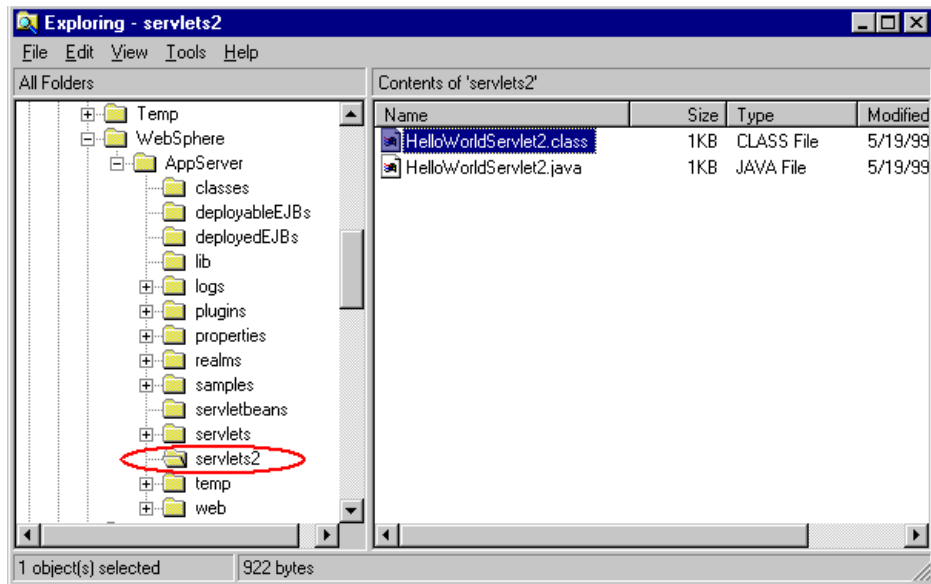


Figure 86. HelloWorldServlet2.class in C:\WebSphere\AppServer\servlets2 Directory

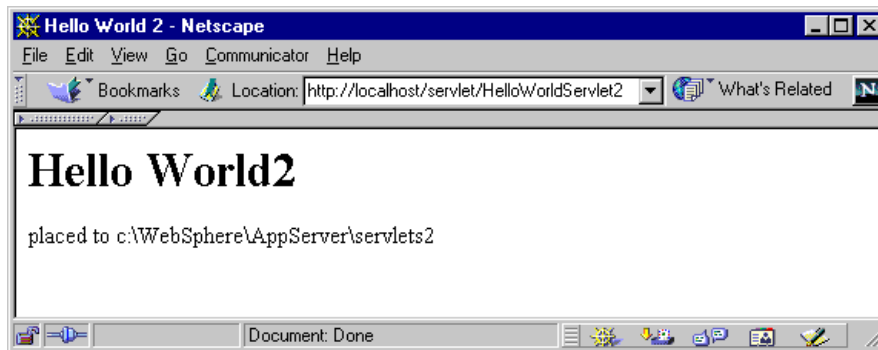


Figure 87. HelloWorldServlet2 Was Invoked from the Alternate Folder

3.1.3.10 How to Configure a Servlet Series (Chaining)

You can create a sequence of servlets in which each one appends additional output to the results of the previous servlet. This is also known as chaining.

The following steps show how to create servlet chains by invoking HelloWorldServlet and snoop from the following Web site:

`http://<hostname>//servlet/ChainServlet`

1. Open the Alias Page by clicking **Servlets -> Aliases** from the Application Server Manager.

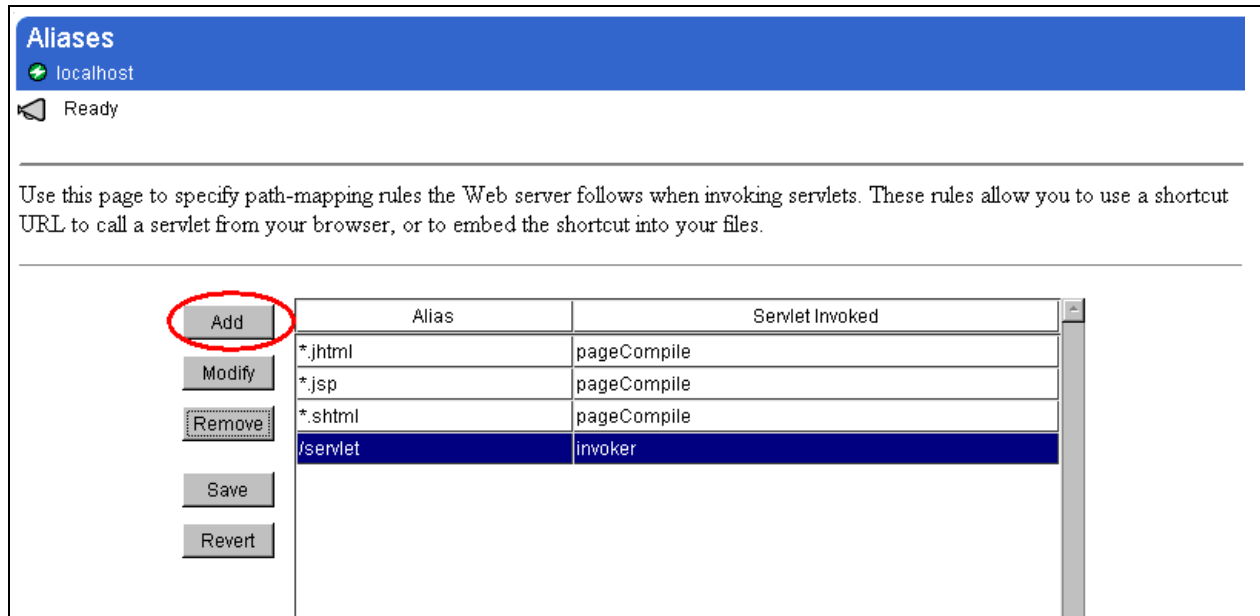


Figure 88. Aliases Page

2. Click the **Add** button. A new row will be added in the table.

Table 7. The Values to Configure Servlet Series

Alias	Servlet Invoked
/servlet/ChainServlet	HelloWorldServlet,snoop (with no space)

3. Update the fields with the correct values (see Figure 89):

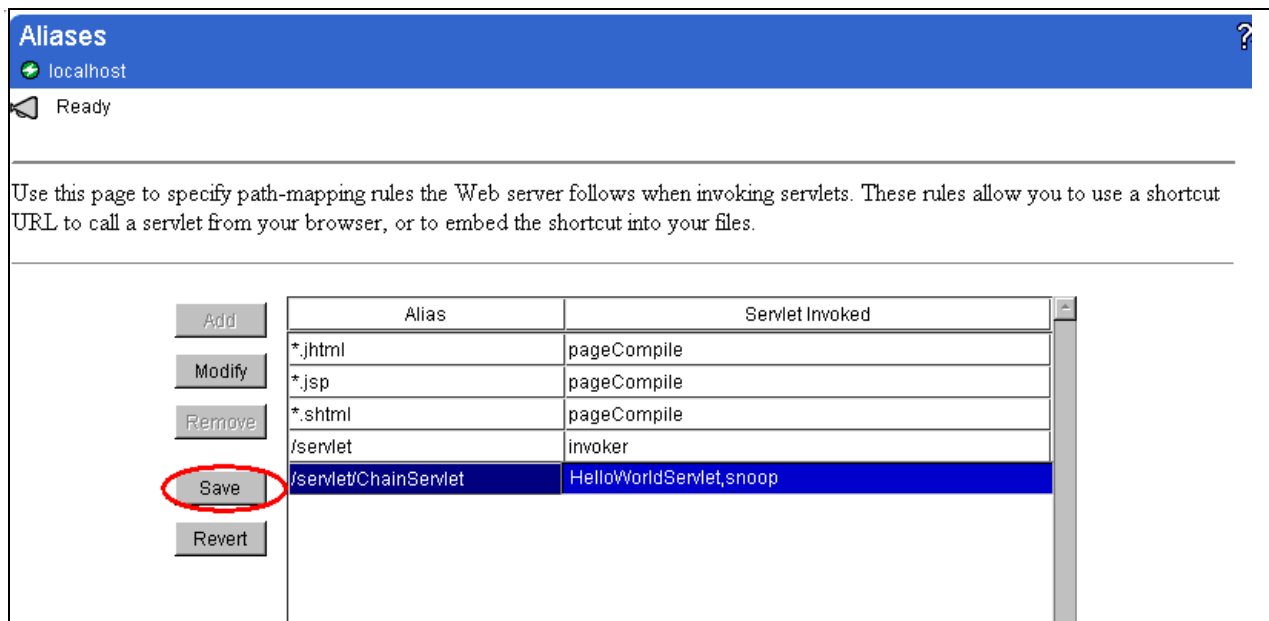


Figure 89. Edit Row Values and Click Save

4. Click the **Save** button (see Figure 89).

5. Open `http://<hostname>//servlet/ChainServlet` with your browser. The following window appears:

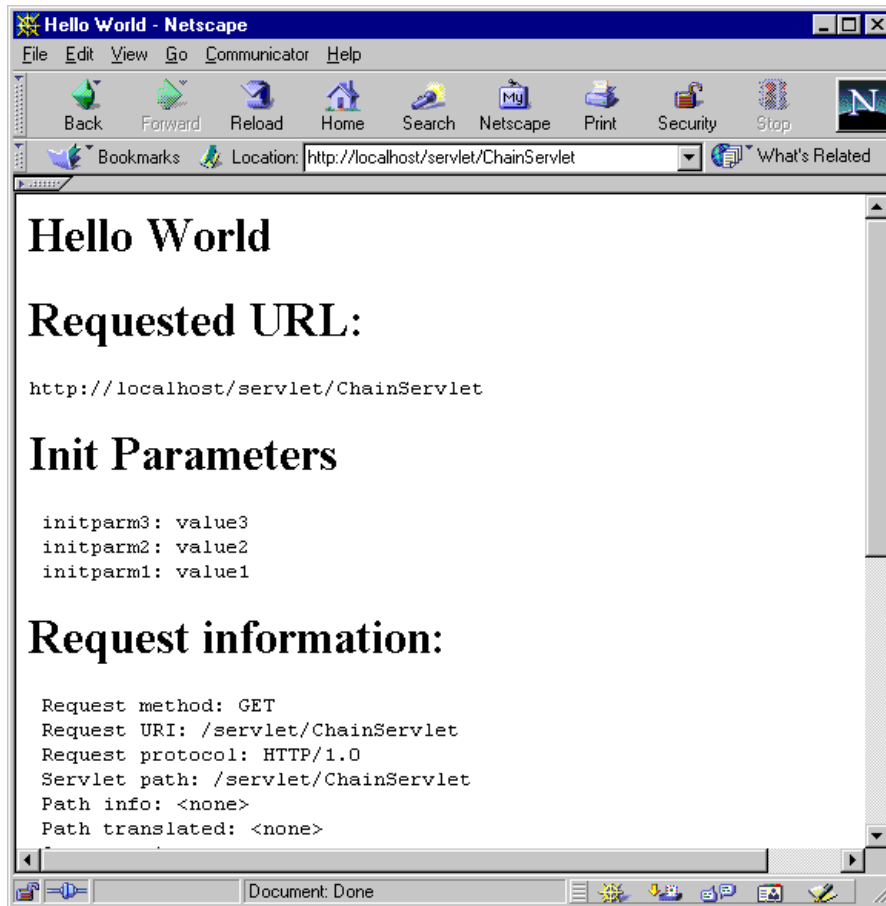


Figure 90. Series Servlets (HelloWorldServlets,snoop)

3.1.3.11 Servlet Filtering

Servlet filtering takes the output of a servlet and uses it as the input of another servlet. This is useful for translation or substitution. For example, you can use this to translate XML to HTML. You can add specific values to your servlet.

Servlet filtering is disabled by default because it uses a lot of resources and causes performance reduction. To enable filtering, do the following:

1. Modify `httpd.properties` in `<ASRoot>/properties/server/servlet/httpd.service`. Change `enable.filters` property from `false` to `true`.

```

#
# Common properties for HTTP and HTTPS services
#
service.vendor=Sun Microsystems
service.version=1.0

# Default HTTP processing pipeline definition:

pipeline.state.class=com.sun.server.http.HttpProcessingState
pipeline.stages=resolver, runner, logger, reporter
pipeline.stage.resolver.class=com.sun.server.http.stages.Resolver
pipeline.stage.runner.class=com.sun.server.http.stages.Runner
pipeline.stage.logger.class=com.sun.server.http.stages.Logger
pipeline.stage.reporter.class=com.sun.server.http.stages.Reporter

# Enable/disable service features
enable.acls=true
enable.browseDirs=false
enable.filters=true
enable.proxy=false
enable.urlLoading=false
enable.virtualHosts=true
:

```

Figure 91. Part of `httpd.properties` (<ASRoot>/properties/server/servlet/httpdbservice)

2. Modify `httpd.properties` in <ASRoot>/properties/servlet/servletservice. Edit the `pipeline.stages` property, that is uncomment the statement for filtering support and comment out the statement that indicates *no* filtering support.

```

#
# Properties for HTTP
#
service.vendor=IBM
service.version=1.1

# Default HTTP processing pipeline definition:
pipeline.state.class=com.sun.server.http.HttpProcessingState

#Stages for servlet processing (no filtering support)
#pipeline.stages=resolver, runner, reporter

#Stages for servlet processing (with filtering support)
pipeline.stages=resolver, filter, runner, reporter

pipeline.stage.resolver.class=com.ibm.servlet.personalization.sessiontracking.IBMResolver
pipeline.stage.runner.class=com.sun.server.http.stages.Runner
pipeline.stage.reporter.class=com.sun.server.http.stages.Reporter
pipeline.stage.filter.class=com.sun.server.http.stages.FilterManager
pipeline.stage.debug.class=com.sun.server.http.stages.Debug
:

```

Figure 92. A Piece of `httpd.properties` (<ASRoot>/properties/servlet/servletservice)

Click the **Servlets -> Filtering** page from the Application Server Manager. This brings up the following window. Click the **Add** button and specify the servlet name and MIME type used to filter (Figure 93 on page 112):

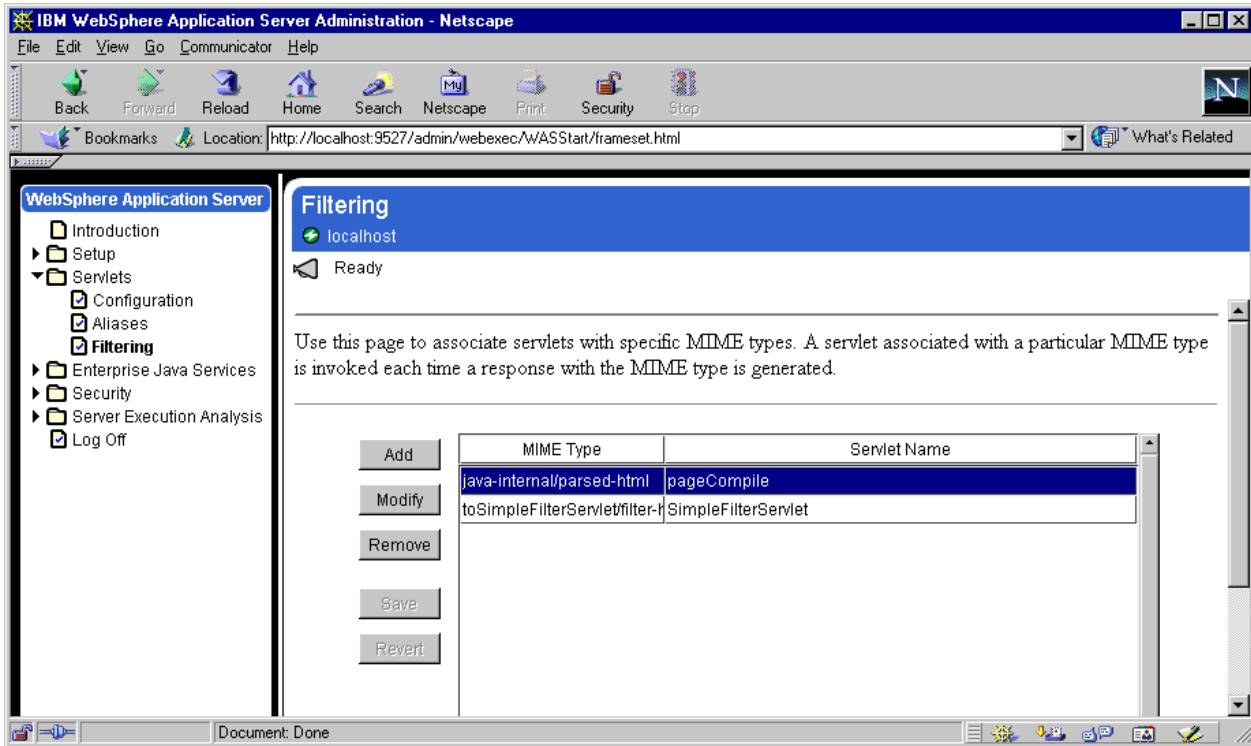


Figure 93. Filtering Setting Page

3. Click **Save**. This change is saved to `<ASRoot>/properties/servlet/servletservice/mimeservlet.properties` file (Figure 94 on page 112):

```
# This file maps mime-types to the servlets which process them
# This is used by the filter manager to set up chains of servlets
# where the output of one servlet gets piped to the input of
# another servlet based on the mime-type that the servlet specifies
# with setContentType("mime-type")
#
# The default servlet for all mime-types is file. Do not set this
# explicitly.
#
# Entries in this file should be of the form
# mime-type/servletname
# ie.
# foo/bar=fooServlet
# where fooServlet is defined in servlets.properties
java-internal/parsed-html=pageCompile
toSimpleFilterServlet/filter-ht=SimpleFilterServlet
```

Figure 94. `mimeservlet.properties`

4. `mimeservlet.properties` is also in `<ASRoot>/properties/servlet/httpservice/`. If you cannot find the MIME type in this file, modify this file and the file shown in Figure 94 on page 112. In that case, the Application Server probably needs to be restarted.
5. Two files are used as filtering samples in this part.
 - `SimpleFilterServlet.java` is the servlet-specified filtering setting. This servlet opens an input stream using `HttpServletRequest.getInputStream()`. See Figure 95 on page 114. Capitalize the characters from the input stream, and write to the output stream.
 - `FilteredServlet.java` is the servlet output HTML code with MIME type specified in the configuration.

```

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

public class SimpleFilterServlet extends HttpServlet {
    public void service(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        try {
            //append a header to the response
            res.setHeader("filtered-by", "SimpleFilterServlet");

            //set the desired content-type of the filtered data
            // res.setContentType("text/plain");
            res.setContentType("text/html");

            //Transfer the header.
            String headersString = "";
            Enumeration e = req.getHeaderNames();
            while (e.hasMoreElements()) {
                String header = (String)e.nextElement();
                String value = req.getHeader(header);
                headersString += header + ": " + value + "\n";
                if (!header.equals("Content-Type")) {
                    res.setHeader(header, value);
                }
            }
            //Warning: getWriter() will flush the stream and thus write the
headers
            //    so we must put this below the set header methods.
            PrintWriter out = res.getWriter();

            //WAS 2.0 and 2.01 had a bug that caused getReader() to throw a
NullPointerException
            //WAS 2.02 has fixed this problem
            BufferedReader in = new BufferedReader(new
InputStreamReader(req.getInputStream()));

            //Xfer and capitalize all of the data from the previous servlet
            String line = in.readLine();
            while(line != null){
                out.println(line.toUpperCase());
                line = in.readLine();
            }
            out.flush();
            out.close();
        }
        catch (Throwable th) {
            th.printStackTrace();
        }
    }
}
} // Thanks to Spike

```

Figure 95. SimpleFilterServlet.java

```

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

public class FilteredServlet extends HttpServlet {

    public void service(HttpServletRequest req, HttpServletResponse res)
        throws IOException {

        res.setContentType("toSimpleFilterServlet/filter-html");
        PrintWriter pw = res.getWriter();
        pw.println("<head>");
        pw.println("<title>FilteredServlet </title>");
        pw.println("<body>");
        pw.println("<h2>This is the output of  FilteredServlet</h2>");
        pw.println("-----<br>");
        pw.println("1234567890<br>");
        pw.println("abcdefghijklmnopqrstuvwxyz<br>");
        pw.println("-----<br>");
        pw.println("</body>");
        pw.flush();
    }
}

```

Figure 96. *FilteredServlet.java*

6. Create `SimpleFilterServlet.java` and `FilteredServlet.java` and compile them.

```

C:\WebSphere\AppServer\Servlets>javac SimpleFilterServlet.java

C:\WebSphere\AppServer\Servlets>javac FilteredServlet.java

C:\WebSphere\AppServer\Servlets>

```

7. Copy the class files into `<ASRoot>/servlets`.

8. Open `http://<hostname>/servlets/FilteredServlet`.

9. If your browser begins downloading(), the MIME setting doesn't work properly. Try restarting the Application Server to enable these settings.

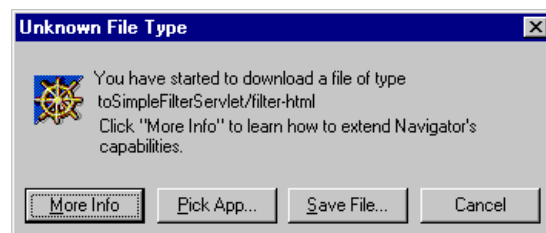


Figure 97. *Download the File*

10. The following window is the result of filtering. All the character output from `FilteredServlet` was capitalized (see Figure 98 on page 116).

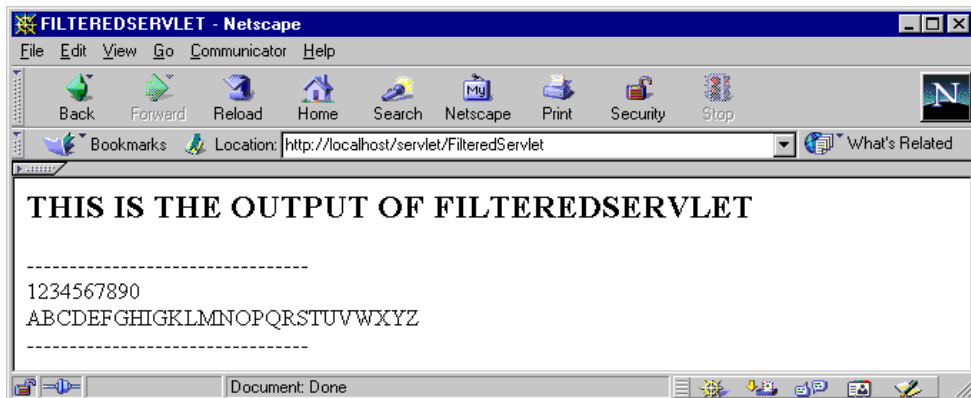


Figure 98. The FilteredServlet Was Filtered and Capitalized by Simple FilterdServlet

3.1.4 Monitoring Servlets

You can see the Loaded Servlets with the Application Server Manager. You don't need any configuration information, as mentioned in 3.1.3, "Configuring a Servlet" on page 96.

From the initial screen, select **Server Execution Analysis -> Monitors -> Loaded Servlets** which leads you to the following window:

Name	Requests	Avg Run Time	State	Auto Start	Auto Reload	Remote	Loaded Time
HelloWorldServlet	0	:00.000	Idle				01-Jun-99 5:48:24 PM HelloWorldSen
SampleBean	0	:00.000	Idle				01-Jun-99 5:48:24 PM SampleBean
snoop	0	:00.000	Idle				01-Jun-99 5:48:24 PM SnoopServlet
UserProfileSample3	0	:00.000	Idle				01-Jun-99 5:48:24 PM UserProfileSan
pageCompile	1	:00.080	Idle				01-Jun-99 5:48:24 PM com.sun.serv
HelloWorldFromRs600	0	:00.000	Idle			✓	01-Jun-99 5:48:30 PM HelloWorldFror
PopulateBeanServlet	1	:00.130	Idle				01-Jun-99 5:48:30 PM PopulateBeanS
ssiServlet	0	:00.000	Idle				01-Jun-99 5:48:30 PM com.sun.serv
invoker	1	:00.130	Idle	✓			01-Jun-99 5:48:36 PM com.sun.serv
CheckMessage	0	:00.000	Idle				01-Jun-99 5:48:40 PM com.ibm.servle

Figure 99. Loaded Servlet Monitor

Table 8. Loaded Servlets Monitor Values

Field	Comment
View	Which servlets to view: servlets you started (user), internal (system) servlets, active servlets, or all of the servlets known to the service.
Name	The servlet name or alias. If a servlet has multiple aliases, more than one row in the table might provide statistics for that servlet. Use the Class Name and Path columns to determine if multiple rows apply to the same servlet.
Requests	The number of requests for the servlet, as received by IBM WebSphere AS since the time the servlet was loaded.
Avg Run Time	The cumulative total number of seconds for all requests to the servlet since it was loaded. The average time a servlet spent in service method such as service(), doGet(), or doPost(). A new average time is calculated each time the Loaded Servlets page is updated. The time is displayed in hh:mm:ss.ms format.
State	The state of the servlet during the current update interval. The possible states are: idle - The servlet is available and is not active. init() - The servlet is initializing. service() - The servlet is performing its services. destroy() - The servlet is stopping. If the servlet is in any state other than idle for many update intervals, suspect a problem with the servlet.
Auto Start	Whether the servlet is configured to load automatically when this service starts (indicated by a check mark) or load when requested (indicated by a blank).
Auto Reload	Whether the servlet is automatically reloaded whenever the service detects a change in the servlet file (indicated by a check mark). The service automatically reloads servlets that are in the monitored directories for servlets and servlet beans. The default monitored paths are <ASRoot>\servlets and <ASRoot>\servletbeans, where <ASRoot> is the directory where Application Server is installed. Internal (system) servlets are not in the monitored paths.
Remote	Whether the servlet was loaded from a remote system (indicated by a check mark) or the local system (indicated by a blank). Remote servlets are loaded in a security sandbox.
Loaded Time	The date and time the servlet was most recently loaded.
Class Name	The servlet class name.
Path	The directory path for the servlet .class file. If the class file is in a JAR file, the path for the JAR file is listed.

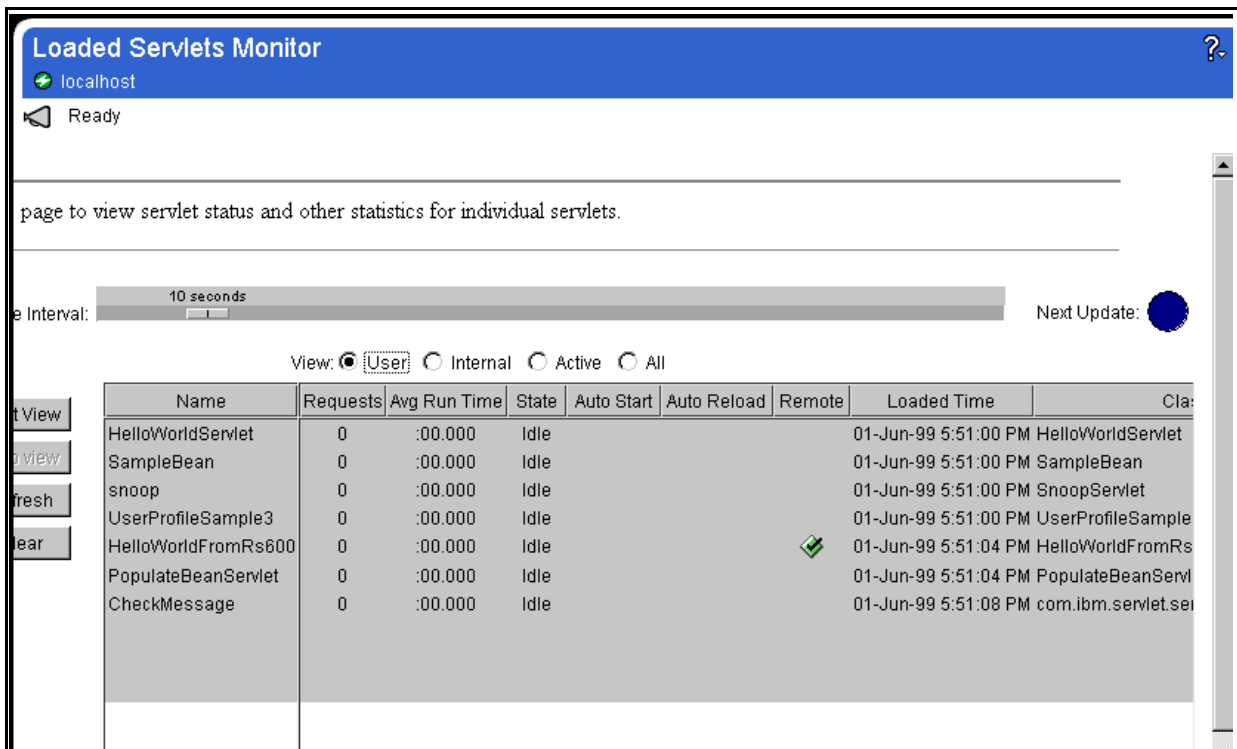


Figure 100. User View Option Checked

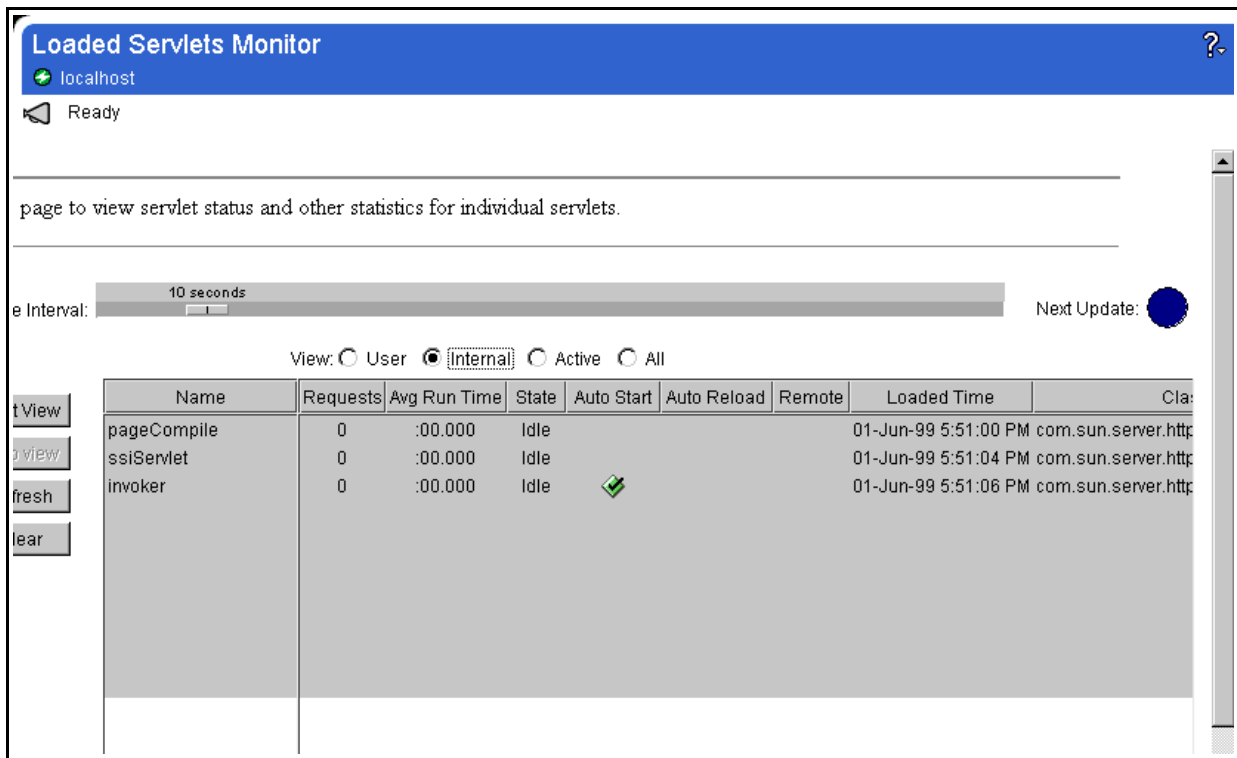


Figure 101. Internal View Option Checked

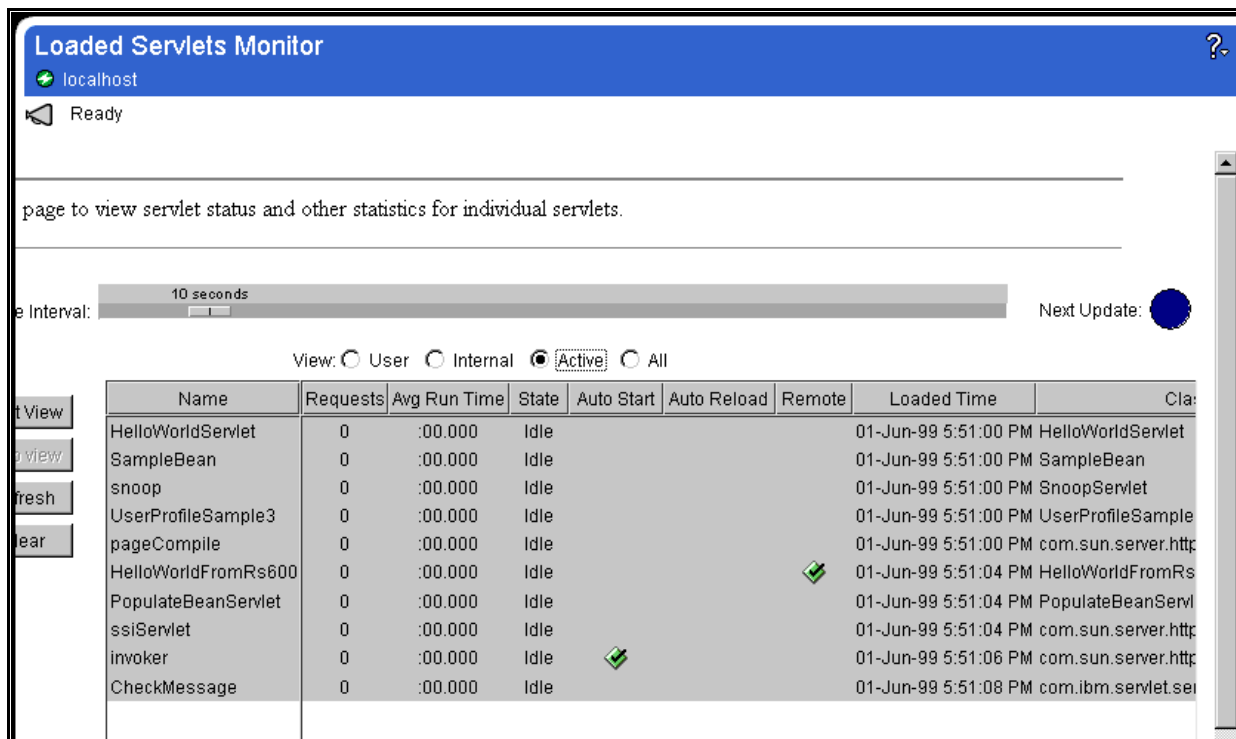


Figure 102. Active View Option Checked

There is an Active View option, but it does not seem to work properly. The state of the monitor is the same as that of the All View option that was checked. This will be fixed in V3.0.

3.2 Java Server Pages

Java Server Pages (JSP) is a relatively new technology developed by Sun Microsystems. It looks like HTML files, but in JSP you can embed the results from servlets and Beans. You can also write Java code in JSP. That enables you to separate the HTML coding from the business logic in your Web applications. WebSphere Application Server implemented JSP before the official announcement and support from Sun Microsystems. When we were writing this part, Sun Microsystems announced JSP V1.0. WebSphere V2.02 used V0.91 of JSP.

For more information, please refer to the following:

- JavaServer Pages (JSP)
[http://\[hostname\]/IBMWebAS/doc/whatis/icjsp.html](http://[hostname]/IBMWebAS/doc/whatis/icjsp.html)
- JavaServer Pages (JSP) reference
[http://\[hostname\]/IBMWebAS/doc/whatis/icjspref.html](http://[hostname]/IBMWebAS/doc/whatis/icjspref.html)
- JavaServer Pages from Sun Microsystems (in the WebSphere documentation)
<http://java.sun.com/products/jsp/>

3.2.1 JSP Architecture

In a JSP environment, most requests are sent to JSP files. Figure 103 on page 121 is a diagram showing the JSP processing flow.

The following list numbers correspond to the numbers in Figure 103 on page 121:

1. The HTTP client sends a request to httpd.
2. JSP sends a request to the servlet instance.

In case there are no servlet instances of the JSP file in JVM:

- The JSP processor(pageCompile) dynamically compiles the JSP file into a Java file and class file (in case files do not exist).
 - The JSP processor generates the servlet instance from the compiled class file.
3. Servlet execute process - send answers back to JSP, or to the client (3').
 4. HTTP client receives the JSP embedded result from the servlet, or the result from the servlet.

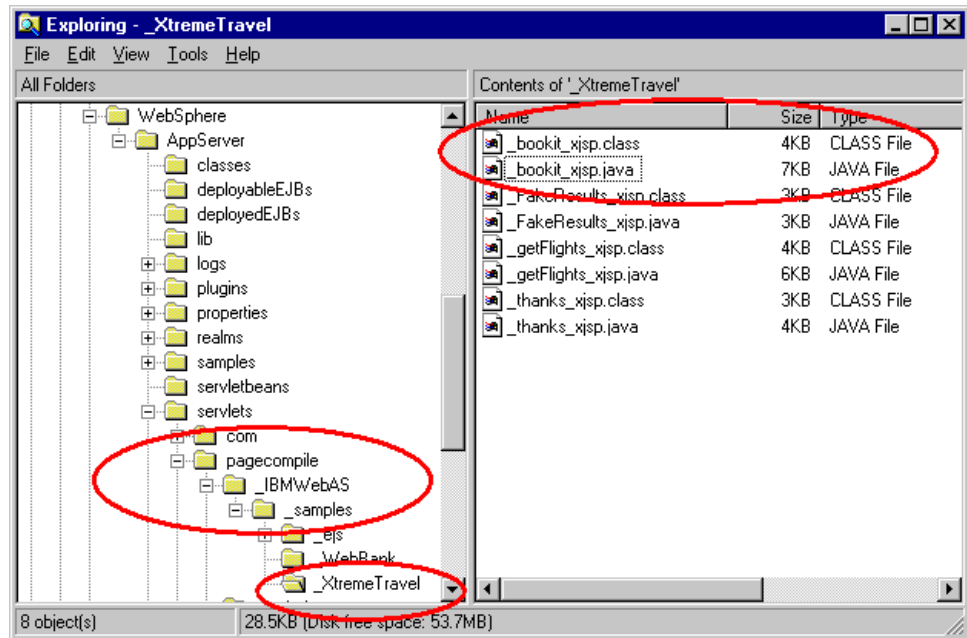


Figure 104. Processed JSP File

The .java and the .class file have the same file name. The processor uses a naming convention that includes adding underscore characters and a suffix to the JSP file name. For example, if the JSP file name is bookit.jsp, the generated files are _bookit_xjsp.java and _bookit_xjsp.class as shown in Figure 104 on page 122.

3.2.1.1 JSP Access Models

There are two ways to access JSP files:

1. *Access Model 1*: Request to JSP, Response from JSP (Figure 105 on page 122).

In this model, Web clients send and receive the requests and responses via a JSP file. A JSP creates servlets and Beans to communicate to back-end systems, if needed.

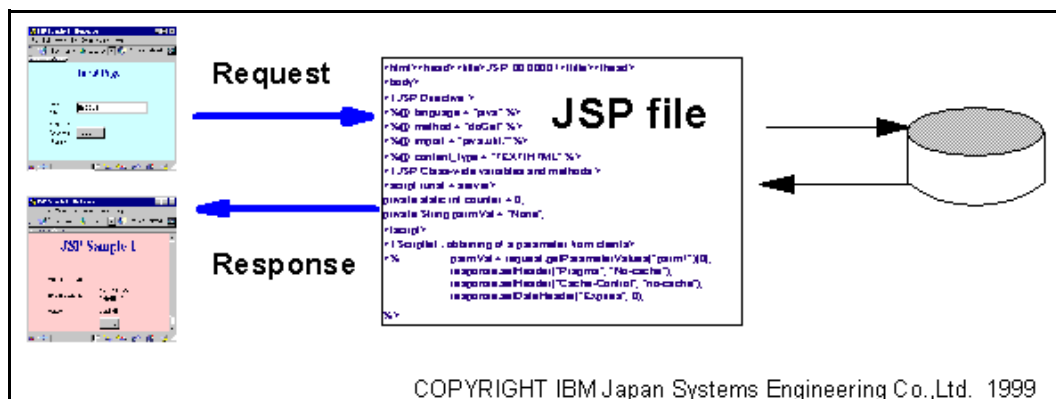


Figure 105. JSP Access Model 1

2. *Access Model 2: Request to Servlet, Response via JSP* (Figure 106 on page 123)

In this model, Web clients send requests to the servlet. The servlet communicates to back-end systems, puts the result into a Java object (usually a Bean), and passes the object to a JSP file. The servlet uses the `com.sun.server.http.HttpServiceResponse` `callPage()` method at that time.

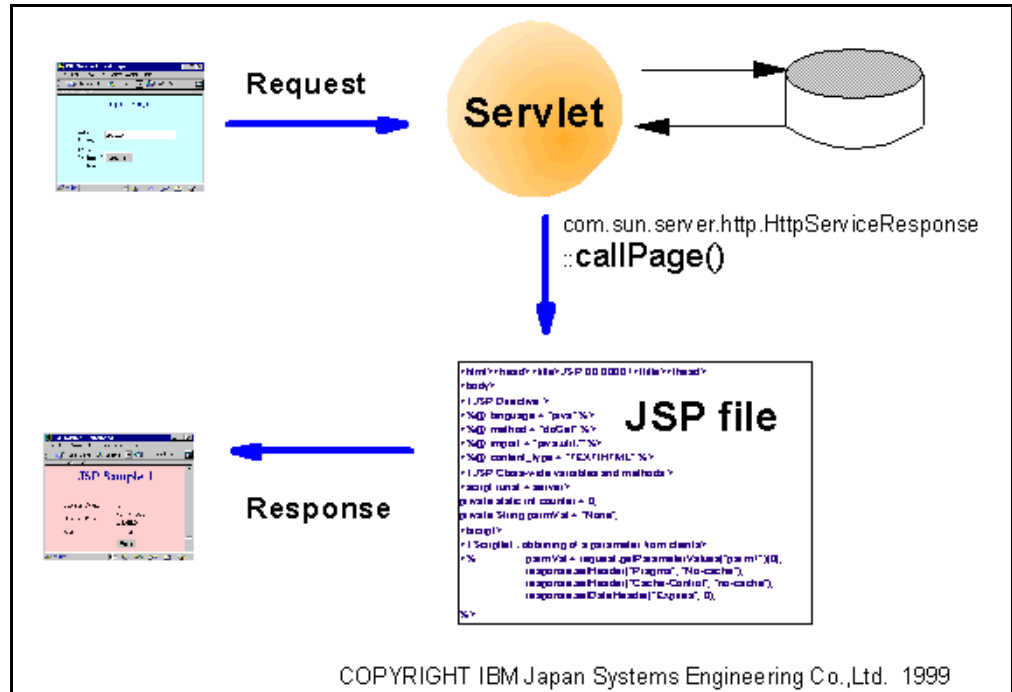


Figure 106. JSP Access Model 2

This approach enables you to separate content generation (business logic) from the presentation of the content (HTML formatting). This separation enables servlets to generate content and store the content (for example, in a Bean) in the request context. The servlet that generated the context generates a response by passing the request context to a JSP file that contains the HTML formatting.

3.2.2 JSP File Contents

JSP files have the extension `.jsp`. A JSP file contains combinations of:

- HTML tags

JSP supports all valid HTML tags. Refer to your favorite HTML reference for a description of those tags.

- NCSA tags

The Application Server supports the following NCSA tags through JSP:

- `config`
- `echo var=variable`
- `exec`
- `filesize`
- `include`
- `lastmodified`

- Commands for formatting size and date outputs
- <SERVLET> tags
See 3.2.3, “<SERVLET> Tags” on page 124.
- JSP Syntax
See the 3.2.4, “JSP Syntax” on page 127.

3.2.3 <SERVLET> Tags

<SERVLET> tags are used to invoke the servlet from JSP. It is useful to embed partial HTML from the servlet to your Web page. The syntax of these tags is:

```
<SERVLET NAME="servlet_name" CODE="servlet_class_name"
CODEBASE="URL_for_remote_loading" initparm1="initparm_value">
```

```
<PARAM NAME="param_name" VALUE="param_value">
```

```
</SERVLET>
```

Table 9. Summary Information of <SERVLET> Tag Attributes

Attributes	Comment
NAME	The servlet class name without .class extension. This can be configured also using the Application Server. See 5.3.1.3, “Creating Multiple Site Queues in a Server” on page 288. You can omit this attribute if CODE attribute is specified. In that case, an instance of the servlet with NAME=CODE is created.
CODE	The servlet class file name. This is optional. See below for tips.
CODEBASE	The URL of the JAR file placed on the remote machine.
parameter	The name and value of initialization parameters.
<PARAM> tag	The name and value of the parameters that are passed through a Request object of the servlet service() method.

Note: The CODE attribute is optional, but when you load a servlet from a remote system, it is not optional. If you drop this attribute, you see a familiar error message (Figure 107 on page 125). If you configure the remote loading of the servlets option on the Servlet Configuration page (in the Application Server Manager), you can load the servlet by specifying the NAME attribute of the <SERVLET> tag. To configure the remote loading of servlets, see 3.1.3.8, “Loading Remote Servlet” on page 103.

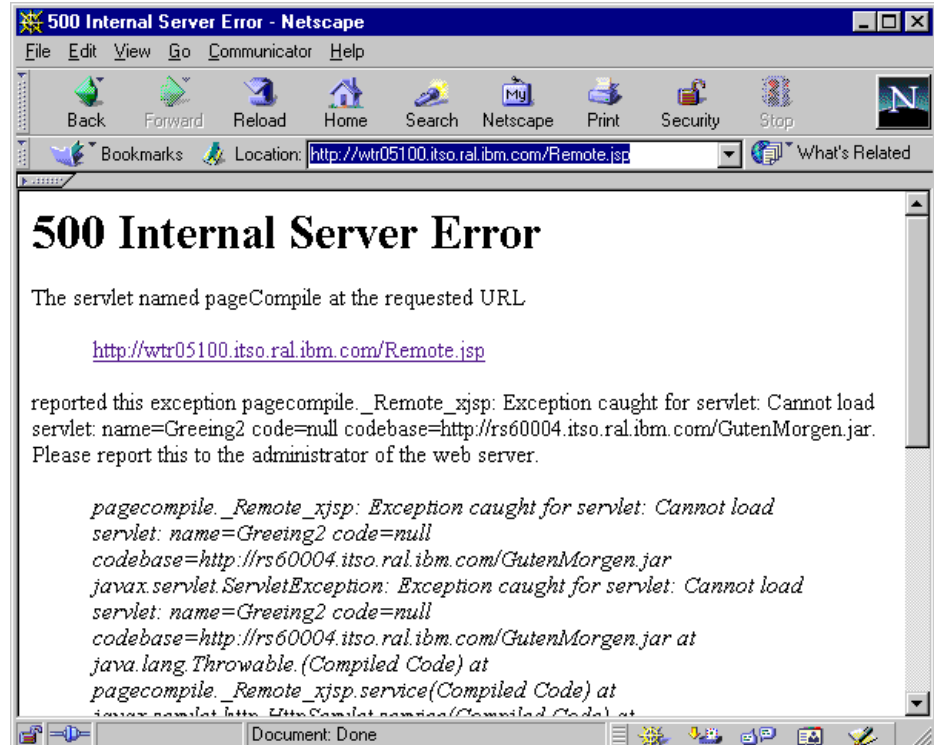


Figure 107. 500 Internal Server Error

This is sample code to show <SERVLET> tags loading a servlet file from the remote system.

```

<html>
<head>
  <title>Remote Servlet</title>
</head>
<body>
<H1>Remote Servlet</H1>
<HR>
<SERVLET NAME="Greeing"
CODE="GutenMorgen.class"
CODEBASE="http://rs60004.itso.ral.ibm.com/GutenMorgen.jar">
</SERVLET>
</body>
</html>

```

Figure 108. Remote.jsp

1. Copy RepeatTest1.jsp to your HTTP document root. In the default setting of IBM HTTP Server1.3.3 on Windows NT, it is located at \Program Files\IBM HTTP Server\htdocs.
2. Create a JAR file from the servlet class file.

```

>jar -cvf GutenMorgen.jar GutenMorgen.class
adding: GutenMorgen.class (in=845) (out=501) (deflated 40%)

```

- Copy GutenMorgen.jar to the HTTP document directory on the remote system (not the servlet root directory). The remote server where you place the .jar file doesn't have to be a WebSphere server.
- The permissions for the JAR file must be set to read for everyone.

```
rs60004:/usr/lpp/HTTPServer/share/htdocs > chmod a+r *.jar
rs60004:/usr/lpp/HTTPServer/share/htdocs > ls -la G*.jar
-rw-r--r-- 1 root      system      927 Jun  2 18:00 GutenMorgen.jar
```

- Open URL `http://<hostname>/Remote.jsp` The following window will appear:

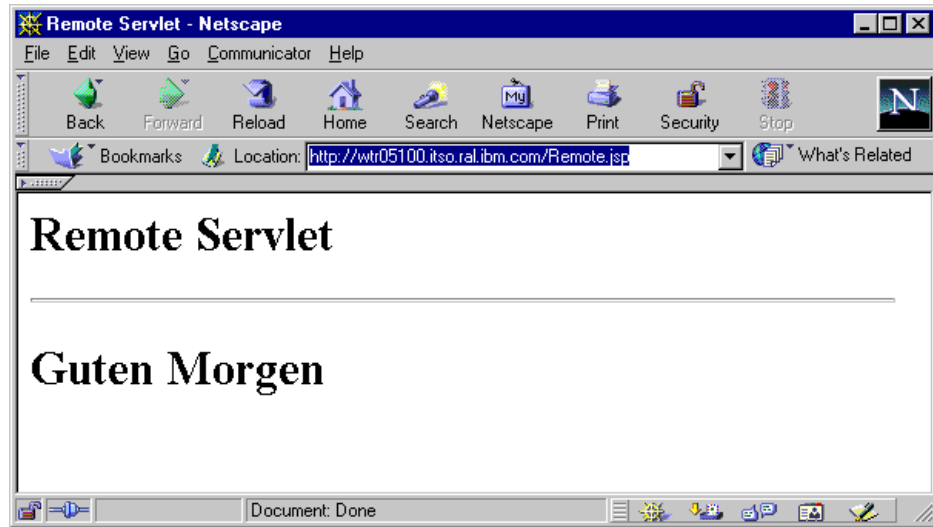


Figure 109. The Result of Remote.jsp

- Click **Servlet Execution Analysis -> Monitors -> Loaded Servlets** and go to the Loaded Servlets Monitor page from the Application Server Manager. You can see that the Greeting instance is created from the GutenMorgen.class (Figure 110 on page 126).

Loaded Servlets Monitor									
localhost									
Ready									
Interval: 10 seconds Next Update: [Refresh]									
View: <input type="radio"/> User <input type="radio"/> Internal <input type="radio"/> Active <input checked="" type="radio"/> All									
Name	Requests	Avg Run Time	State	Auto Start	Auto Reload	Remote	Loaded Time	Class Name	Path
pageCompile	1	:01.152	Idle				02-Jun-99 6:08:08 PM	com.sun.server.http.pagecompile.PageCom	C:\WebSphere\AppServer\lib\st.jar
Greeting	1	:00.010	Idle			✓	02-Jun-99 6:08:20 PM	GutenMorgen.class	http://rs60004.itso.ral.ibm.com/GutenMorge
ssiServlet	0	:00.000	Idle				02-Jun-99 6:08:16 PM	com.sun.server.http.ssi.ServerSideInclude	C:\WebSphere\AppServer\lib\st.jar
invoker	0	:00.000	Idle	✓			02-Jun-99 6:06:36 PM	com.sun.server.http.InvokerServlet	C:\WebSphere\AppServer\lib\st.jar

Figure 110. The Greeting Instance Was Created

The sample `<SERVLET>` tags are also in 5.3.1.2, “Showing Bulletin” on page 287 and 5.3.2.1, “Sample Model of Message Exchanging” on page 296.

You can find detailed explanations about <SERVLET> tags at:

<http://<hostname>//IBMWebAS/doc/howto/itinvsv.html#invshhtml>

3.2.4 JSP Syntax

JSP syntax consists of the following format:

- <%@ and %> Tags (JSP Directives)
- <SCRIPT> and </SCRIPT> Tags
- <% and %> Tags (Inline Java Code, Scriptlets)
- <%= and %> Tags (Java Expressions for Variable Data)
- <BEAN> Tag (Accessing JavaBeans)
- <INSERT> Tag (For Embedding Variables in an HTML Page)
- <REPEAT> Tags (For Repeating a Block of HTML tagging)

You can get detailed information about JSP syntax from

<http://<hostname>/IBMWebAS/doc/whatis/icjspref.html>.

The references in the WebSphere documentation are located at:

<http://www.javasoft.com/products/jsp/index.html>

You can also get JSP information from:

http://www.javasoft.com/products/jsp/JSP-1_0-public-draft-1.pdf

3.2.4.1 <%@ and %> Tags (JSP Directives)

In Figure 111 on page 128, you can find the expressions enclosed within <%@ and %>. This syntax is called *JSP directives*. JSP directives are used to specify:

- The scripting language being used
- The interfaces a servlet implements
- The classes a servlet extends
- The packages a servlet imports

The general syntax of the JSP directives is:

```
<%@ directive_name = "value" %>
```

In JSP directives, the following six directives are valid as directive names:

language, method, import, content_type, implements, extends

With JSP directives, you define basic settings (for example, language or method) for the code generated from JSPs:

- Language

The language directive defines the scripting language used in this file. At this time, only *Java* is valid and it is set as the default.

- Method

This is the method of servlet code generated from a JSP file. The default method is *service*.

- **Import**

You can import Java language packages just like writing a servlet. This directive can be specified multiple times within a JSP file to import different packages.

- **Content_type**

This is the MIME type of the generated response. The default value is *text/html*.

- **Implements**

This is a comma-separated list of Java language interfaces that the generated servlet implements. You can use this directive more than once within a JSP file to implement different interfaces.

- **Extends**

This is the name of the Java language class that the servlet extends. The class must be a valid class and does not have to be a servlet class. The scope of this directive spans the entire JSP file. When used more than once, only the first occurrence of the directive is significant.

The following code segment provides an example of JSP directives. You usually use JSP directives in the beginning part of your code.

```
<html><head>
<title>JSP Sample for JSP Directive </title></head>
<body text="#000000" bgcolor="#FFCCCC" link="#0000EE" vlink="#551A8B"
alink="#FF0000">

<! JSP Directives >
<%@ language = "java" %>
<%@ method = "doGet" %>
<%@ import = "java.util.*"%>
<%@ import = "java.sql.*"%
<%@ implements = "javax.servlet.http.HttpSessionContext" %>
<%@ content_type = "text/html" %>
<%@ extends = "javax.servlet.http.HttpServlet" %>
```

Figure 111. JSP Directives

3.2.4.2 <SCRIPT> and </SCRIPT> Tags

Use the <SCRIPT> and </SCRIPT> tags to declare class-wide variables and class-wide methods for the servlet class. The general syntax is:

```
<script runat=server>
// code for class-wide variables and methods
</script>
```

The attribute `runat=server` is required and indicates that the tag is for server-side processing. You shouldn't edit this expression.

```

<! JSP Class-wide variables and methods>
<script runat = server>
private static int counter = 0;
private String parmVal = "None";
private String arrangeCreatedTime ( GregorianCalendar gcl) {
GregorianCalendar gcal = gcl;
StringBuffer arrangedTime = new StringBuffer();
arrangedTime.append("<B>" + (1+gcal.get(Calendar.MONTH)));
arrangedTime.append("/") + gcal.get(Calendar.DATE));
arrangedTime.append("/ " + gcal.get(Calendar.YEAR));
arrangedTime.append("<BR>" +gcal.get(Calendar.HOUR_OF_DAY));
arrangedTime.append(": " +gcal.get(Calendar.MINUTE));
arrangedTime.append("</B>:" +gcal.get(Calendar.SECOND));
return arrangedTime.toString();
}
</script>

```

Figure 112. <SCRIPT> Tag

3.2.4.3 <% and %> Tags (Inline Java Code, Scriptlets)

You can embed any valid Java language code inline within a JSP file between the <% and %> tags. An advantage of embedding Java coding for servlets inline in JSP files is that the servlet does not have to be compiled in advance, and placed on the server. This makes it easier to quickly test servlet coding. You can use these variables:

- request, responses, out, in

These are pre-defined for you. They are closely related to the classes well known to Servlet developers. The classes are `HttpServletRequest`, `HttpServletResponse` in `javax.servlet.http` package, `PrintWriter` and `BufferedReader` in `java.io` package.

- The variables that you defined in <SCRIPT> tag (see 3.2.4.2, “<SCRIPT> and </SCRIPT> Tags” on page 128).

In this code, you get a parameter value of *parm1* (this is the string that you input in the text field) from the HTTP request and set HTTP headers to disable caches as shown in Figure 127 on page 143.

3.2.4.4 <%= and %> Tags (Java Expressions for Variable Data)

The tags, <%= and %>, are used to embed the value of variable data to the JSP. Primitive types, such as *int* and *float*, are automatically converted to a string representation.

This is an example:

- <%= (++counter)%>

This increments the counter variable and displays the value.

- <%= arrangeCreatedTime(new GregorianCalendar())%>

This executes the `arrangeCreatedTime()` method and displays the value. You can define this method in the <SCRIPT> tag.

As you can see, you can execute simple calculations and methods in this tag.

3.2.4.5 <BEAN> Tag (Accessing JavaBeans)

This tag is used to access JavaBeans. When you want to access Beans, you must declare the Bean to be used. With this tag, you can get the value from the Bean. The <BEAN> tag syntax is:

- name

```
<bean name="Bean_name" varname="local_Bean_name"
      type="class_or_interface_name" introspect="yes|no"
      beanName="ser_filename" create="yes|no"
      scope="request|session|userprofile" >
  <param property_name="value">
</bean>
```

Name is used to look up the Bean in the appropriate scope (specified by the scope attribute). The value is case-sensitive. For example, when using the Bean from PopulateBeanServlet.java to DisplayData.jsp, this value must be the same name as that stored in the setAttribute() method as shown in the following window:

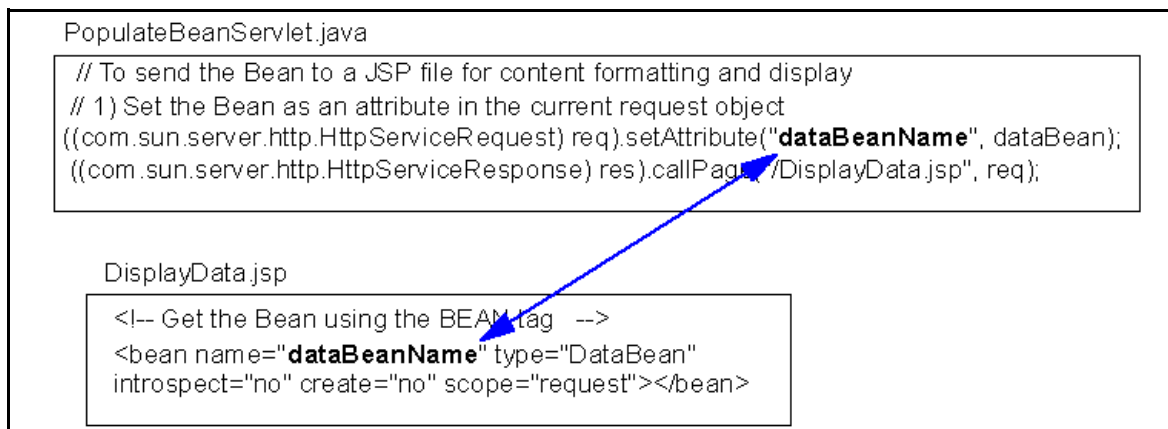


Figure 113. These Two Data Bean Names Should Be the Same Value

- varname

This attribute is optional. When you use the Bean elsewhere in JSP, you refer to the Bean with this value.

- type

The name of the Bean class name.

- introspect

When the value is yes, the JSP processor examines all requested properties and calls the set property methods (passed in the BeanInfo) that match the requested properties. The default value of this attribute is yes.

- beanName

The serialized object file name (.ser file), or the name of the Bean .class file, the Bean package name.

- create

When the value is yes, the JSP processor creates an instance of the Bean if the processor does *not* find the Bean within the specified scope. The default

value is yes. In DisplayData.jsp (Figure 113 on page 130), if you change this value to *no*, the result is the same, because the processor finds the Bean within the scope.

- scope

This is the lifetime of the Bean. This attribute is optional. There are three valid values:

- request

This is the default value. The Bean is set as a context in the request by a servlet that invokes the JSP file using the APIs described in the JSP API. That is, the life time of the bean is one HTTP session.

If the Bean is not part of the request context, the Bean is created and stored in the request context, unless the create attribute is set to no.

- session

If the Bean is present in the current session, the bean is reused. If the Bean is not present, it is created and stored as part of the session (if the create attribute is set to yes).

- userprofile

The user profile is retrieved from the servlet request object, put to the specified type and introspected. If a type is not specified, the default type is `com.ibm.servlet.personalization.userprofile.UserProfile`. The create attribute is ignored.

- param

A list of property and value pairs. The properties are automatically set in the bean using introspection. The properties are set once when the Bean is instantiated.

3.2.4.6 <INSERT> Tag

The <INSERT> tag is used to embed the values of the Java variable into JSP.

The general syntax is:

```
<insert requestparm=pvalue requestattr=avalue bean=name  
  property=property_name(optional_index).subproperty_name(optional_index)  
  default=value_when_null>  
</insert>
```

Where:

- *requestparm*

The parameter that is accessed in the request object. This attribute is case-sensitive and can't be used with the Bean and property attributes.

- *requestattr*

The attribute that is accessed in the request object. The attribute is set with the `setAttribute` method. This attribute is case-sensitive and can't be used with the Bean and property attributes.

- *bean*

The name of the JavaBean declared by a <BEAN> tag within the JSP file. For more information on the <BEAN> tag, see 3.2.4.5, “<BEAN> Tag (Accessing JavaBeans)” on page 130. The value for this attribute is case-sensitive.

When the Bean attribute is specified, but the property attribute is not specified, the entire Bean is used in the substitution.

For example, if the property is not specified, the String of the Bean value is displayed as the substitution.

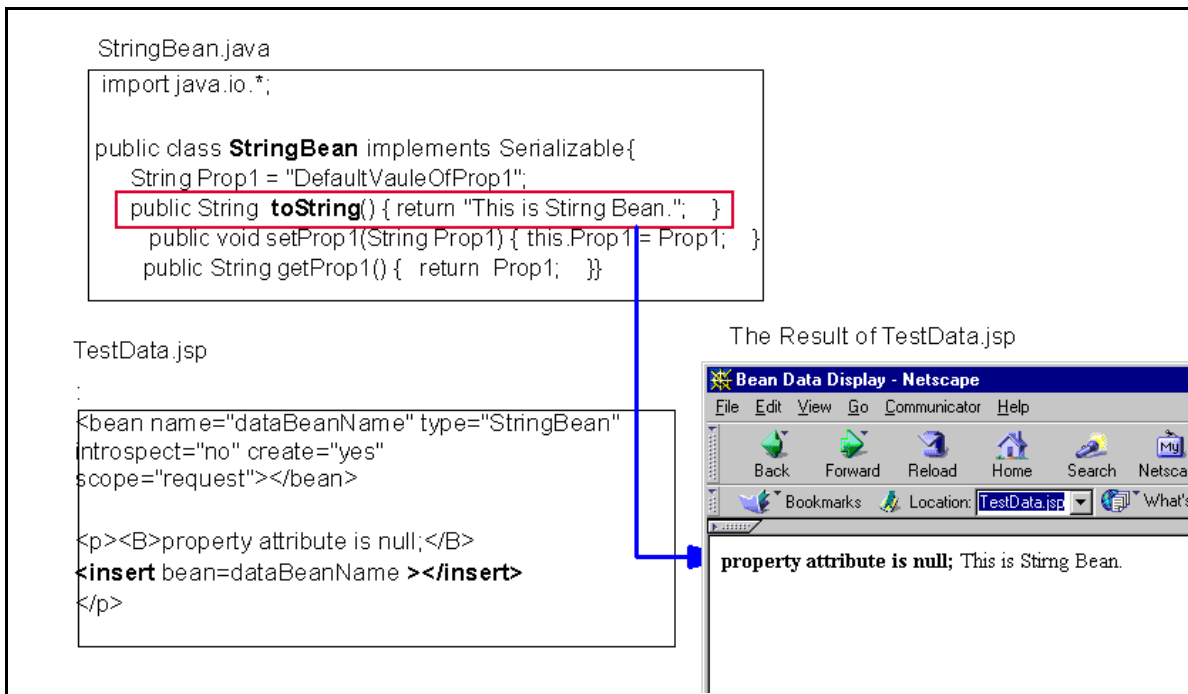


Figure 114. The Bean Is Used in the Substitution, If the Property Is Not Specified

- *property*

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. This attribute can't be used with the requestparm and requestattr attributes. For example, the code snippet in Figure 115 on page 133 displays the value of prop3 for the DataBean.


```

<bean name="dataBeanName" type="DataBean" introspect="no" create="yes"
scope="request">
</bean>
<body>
:
:
<p>The value of Bean property 3 is
<insert bean=dataBeanName property=prop3 default="No property value" >
</insert></p>

```

Figure 115. Example of <INSERT> Tag

```

The value of Bean property 3 is Value3

```

Figure 116. The Result of Figure 115

- *default*

An optional string to display when the value of the bean property is null. The default value of this parameter is an empty string.

3.2.4.7 <REPEAT> Tags (For Repeating a Block of HTML Tagging)

Use <REPEAT> tags to repeat a block of HTML tags that contains the <INSERT> tag and the HTML tags for formatting the content.

The syntax of the <REPEAT> tags is:

```

<repeat index=name start=starting_index end=ending_index>
</repeat>

```

where:

- *index*

An optional name used to identify the index for this repeat block. The value is case-sensitive.

- *start*

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

- *end*

An optional ending index value for this repeat block. The maximum value is 2,147,483,647. If the value for the end attribute is less than the value of the start attribute, the end attribute is ignored.

<REPEAT> tags are useful to access the indexed properties of JavaBeans.

RepeatTestBean.java (Figure 117 on page 135) is an example of properties that are indexed. RepeatTest1.jsp accesses the RepeatTestBean with <REPEAT> tags. RepeatTest1.jsp is one file, but it is cut into pieces here because it is too

large to be shown at once and because we want to make it clear what each segment of HTML code does.

To use this sample:

1. Create RepeatTestBean.java and RepeatTest1.jsp.
2. Copy RepeatTest1.jsp to your HTTP document root. In the default setting of IBM HTTP Server1.3.3 on Windows NT, it is located at \Program Files\IBM HTTP Server\htdocs.
3. Copy RepeatTestBean.java into <ASRoot>/servlets. Compile RepeatTestBean.java and create a class file.

```
C:\WEBSPH~1\APPSER~1\servlets>javac RepeatTestBean.java
```

```
C:\WEBSPH~1\APPSER~1\servlets>
```

4. You can get the result by opening the URL `http://<hostname>/RepeatTest1.jsp`.

```

import java.io.*;

public class RepeatTestBean implements Serializable{
    String[] city = new String[5];
    String[] address = new String[5];
    String[] telephone = new String[5];

    public static void main(String[] args){
        RepeatTestBean b=new RepeatTestBean();
        for(int i=0; i<5; i++){
b.setCity(i,"cityName"+i);
b.setAddress(i,"AddressValue"+i);
b.setTelephone(i,"telephoneNumber"+i);
        }
        for(int i=0; i<3; i++){
            System.out.println("city: "+b.getCity(i));
            System.out.println("add : "+b.getAddress(i));
            System.out.println("tel : "+b.getTelephone(i));
        }
    }
    public String toString() {
        return "This is RepeatTestBean.";
    }
    //----- city -----//
    public void setCity(int index,String city) {
        this.city[index] = city;
    }
    public void setCity(String[] city) {
        this.city = city;
    }
    public String[] getCity() {
        return city;
    }
    public String getCity(int index) {
        return getCity()[index];
    }
}

```

Figure 117. RepeatTestBean.java (1/2)

```

//----- address -----//
public void setAddress(int index,String address) {
    this.address[index] = address;
}
public void setAddress(String[] address) {
    this.address = address;
}
public String[] getAddress() {
    return address;
}
public String getAddress(int index) {
    return getAddress()[index];
}
//----- telephone -----//
public void setTelephone(int index,String telephone) {
    this.telephone[index] = telephone;
}
public void setTelephone(String[] telephone) {
    this.telephone = telephone;
}
public String[] getTelephone() {
    return telephone;
}
public String getTelephone(int index) {
    return getTelephone()[index];
}
}

```

Figure 118. RepeatTestBean.java (2/2)

```

<html>
<head>
<title>RepeatTest1.jsp</title>
</head>
<H1>Test of Repeat Tags</H1>
<!-- Get the Bean using the BEAN tag -->
<bean name="serviceLocationsQuery" type="RepeatTestBean" introspect="no"
create="yes" scope="request">
</bean>
<body>

<!--- scriptlet -- >
<%
for(int i=0; i<5; i++){
  serviceLocationsQuery.setCity(i, "cityName"+i);
  serviceLocationsQuery.setAddress(i, "AddressValue"+i);
  serviceLocationsQuery.setTelephone(i, "telephoneNumber"+i);
}
%>
<HR>
<H2> Repeat Tags Example1</H2>
<table border=2>
<tr><td><B>city</B></td><td><B>Address</B></td><td><B>Telephone</B></td></
tr>
<repeat>
<tr><td><insert bean=serviceLocationsQuery property=city></insert></td>
<td><insert bean=serviceLocationsQuery property=address></insert></td>
<td><insert bean=serviceLocationsQuery
property=telephone></insert></td></tr>
</repeat>
</table>

```

Figure 119. RepeatTest1.jsp (1/3)

RepeatTest1.jsp accesses the RepeatTestBean three different ways using the <REPEAT> tags. In the beginning of this file, enclosed with bold <% and %> in Figure 119 on page 137, it sets the values to the properties of the RepeatTestBean.

Figure 120 on page 138 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 will repeat.

The result of this part is shown in Figure 120 on page 138. All elements of the properties are restored with this tag.



Figure 120. The Result of RepeatTest1.jsp (2/3)

Figure 121 shows the starting index and the ending indexes:

```

<HR>
<H2> Repeat Tags Example2</H2>
<table border=2>
<tr><td><B>city</B></td><td><B>Address</B></td><td><B>Telephone</B></td></tr>
<repeat index=myIndex1 start=2 end=2147483647>
<tr><td><insert bean=serviceLocationsQuery
property=city(myIndex1)></insert></td>
<td><insert bean=serviceLocationsQuery
property=address(myIndex1)></insert></td>
<td><insert bean=serviceLocationsQuery
property=telephone(myIndex1)></insert></td></tr>
</repeat>
</table>
*

```

Figure 121. RepeatTest1.jsp (3/3)

The result of this part is shown in Figure 122 on page 139. The start attribute for the <REPEAT> tags and the values for the properties begin at XXX2.

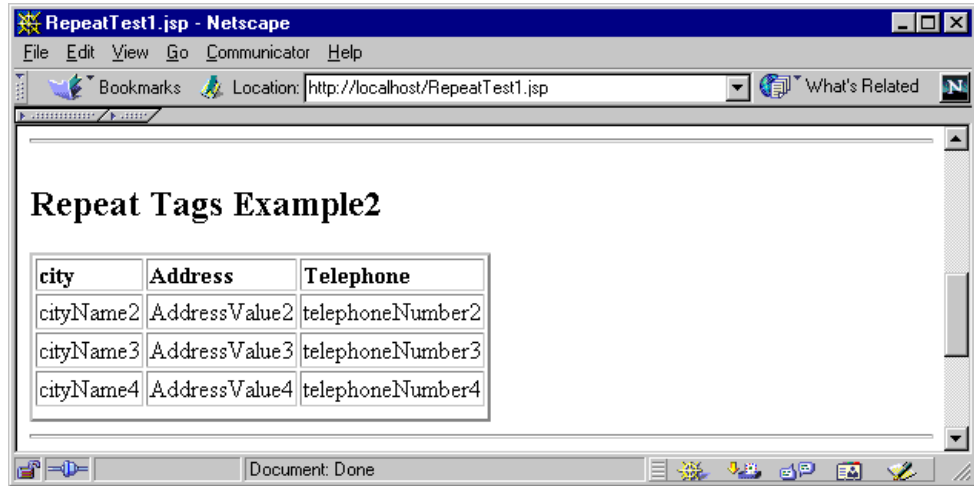


Figure 122. The Result of RepeatTest1.jsp

Figure 123 shows explicit indexing and ending indexing with an implicit starting index. Although the index attribute is specified, the indexed property *city* can still be implicitly indexed because the value (i) is not required.

```

<HR>
<H2> Repeat Tags Example3</H2>
<table border=2>
<tr><td><B>city</B></td><td><B>Address</B></td><td><B>Telephone</B></td></tr>
<tr>
<td><insert bean=serviceLocationsQuery property=city></insert></td>
<td><insert bean=serviceLocationsQuery
property=address (myIndex2) ></insert></td>
<td><insert bean=serviceLocationsQuery
property=telephone (myIndex2) ></insert></td></tr>
</repeat>
</table>
</body></html>

```

Figure 123. RepeatTest1.jsp

Figure 124 on page 140 shows you the result. The display of the value of the properties stopped at XXX2 because of the end property <REPEAT> tags. Previous examples displayed all of the elements.

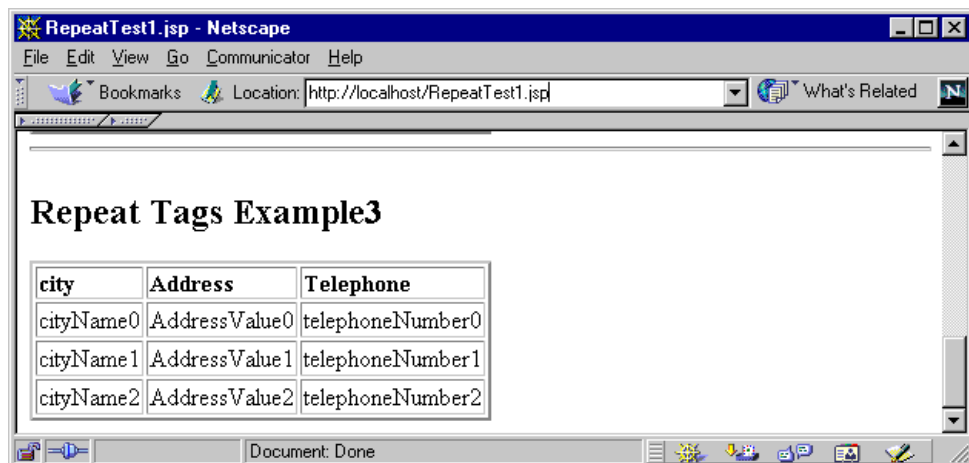


Figure 124. The Result of RepeatTest1.jsp

3.2.5 JSP APIs

Two interfaces are provided to pass the request context to a JSP file. The interfaces that support JSP are:

1. `com.sun.server.http.HttpServletRequest`
 This class implements the `javax.servlet.http.HttpServletRequest` interface and a `setAttribute()` method to set attributes defined by name.
2. `com.sun.server.http.HttpServletResponse`
 This class implements the `javax.servlet.http.HttpServletResponse` interface and adds a `callPage()` method enabling servlets to call JSP files and optionally pass a context.

3.2.5.1 callPage() method

Use the `callPage()` method to serve a JSP from within a servlet. The served page (a JSP file) is returned as the response to the browser. The calling servlet can also pass some context via the request object. You should code the header of the served page to include a directive to tell the browser not to cache the file.

The syntax of the `callPage()` method is:

```
public void callPage(String fileName,  
HttpServletRequest req) throws ServletException, IOException)
```

Where:

- *fileName*

Is the name of the URL that identifies the file that will be used to generate the output and present the content. If the file name begins with slash (/), the file location is assumed to be relative to the document root. If the file name doesn't begin with a slash, the location is assumed to be relative to the URL with which the current request was invoked.

The `callPage()` method doesn't support calling pages with the file extension `.html`. If you need to invoke HTML pages using the `callPage()` method, you must first rename the HTML files to have the file extension `.jsp`.

- *req*

Is the `HttpServletRequest` object of the servlet that invoked this method. Most often, the content is passed as a Bean in the context of the request object.

To use the `callPage()` method, you must pass the response object to a special Sun object:

com.sun.server.http.HttpServiceResponse.

See Figure 137 on page 149 for an example.

3.2.5.2 `setAttribute()` method

Use the `setAttribute()` method to store an attribute in the request context. The syntax is:

```
public void setAttribute(String key, Object o)
```

Where:

- *key*

Is the name of the attribute to be stored.

- *o*

Is the context object stored with the key.

To use the `setAttribute()` method, you must pass the request object to a special Sun object:

com.sun.server.http.HttpServiceRequest.

See Figure 137 on page 149 for an example of the syntax.

3.2.6 JSP Sample1

We show a very simple JSP sample first. This is a sample of the access model 1, shown in 3.2.1.1, “JSP Access Models” on page 122.

3.2.6.1 Sample1 Processing Flow

We show you two files, `Sample1.html` (Figure 126 on page 142) and `Sample1.jsp`. Parameters are sent to `Sample1.jsp` (Figure 127 on page 143), which processes the parameter and sends the response to the browser as shown in Figure 125 on page 142:

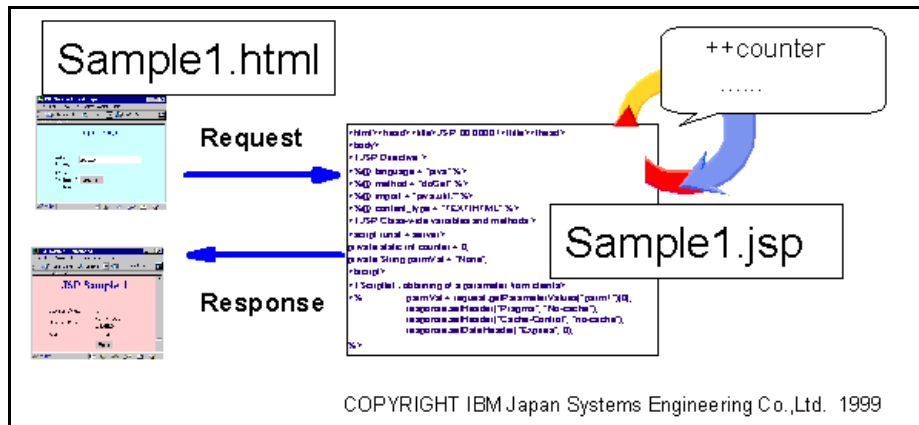


Figure 125. JSP Sample1 Processing Flow

```

<html><head>
  <title>JSP Sample 1</title>
</head>
<body text="#000000" bgcolor="#CCFFFF" link="#0000EE" vlink="#551A8B"
alink="#FF0000">

<center><font color="#000099"><font size=+2>Input
Page</font></font></center>

<form action = "http://localhost/Sample1.jsp" method = "GET">
<p><br>
<center><table COLS=2 WIDTH="76%" >
<tr>
<td>Input String:</td>

<td><input type = "text" name = "parm1"></td>
</tr><tr><td>Click on "Submit!" Button:</td>
<td><input type = "submit" VALUE = "Submit!"></td>
</tr></table></center>
</form></body></html>

```

Figure 126. Sample1.html

```

<html><head>
<title>JSP Sample1</title></head>
<body text="#000000" bgcolor="#FFCCCC" link="#0000EE" vlink="#551A8B"
alink="#FF0000">

<! JSP Directive >
<%@ language = "java" %>
<%@ method = "doGet" %>
<%@ import = "java.util.*"%>
<%@ content_type = "TEXT/HTML" %>

<! JSP Class-wide variables and methods>
<script runat = server>
private static int counter = 0;
private String parmVal = "None";
private String arrangeCreatedTime ( GregorianCalendar gcl) {
GregorianCalendar gcal = gcl;
StringBuffer arrangedTime = new StringBuffer();
arrangedTime.append("<B>" + (1+gcal.get (Calendar.MONTH)));
arrangedTime.append("/" + gcal.get (Calendar.DATE));
arrangedTime.append("/ " + gcal.get (Calendar.YEAR));
arrangedTime.append("<BR>" +gcal.get (Calendar.HOUR_OF_DAY));
arrangedTime.append(": " +gcal.get (Calendar.MINUTE));
arrangedTime.append("</B>:" +gcal.get (Calendar.SECOND));
return arrangedTime.toString();
}
</script>

<! Scriptlet : disabling caches>
<%
parmVal = request.getParameterValues("parm1")[0];
response.setHeader("Pragma", "No-cache");
response.setHeader("Cache-Control", "no-cache");
response.setDateHeader("Expires", 0);
%>

<center>
<H1><font color="#000099">JSP Sample 1</font></H1>
</center>
<form action = "http://localhost/wakako/Sample1.html" method = "GET">
<p><br>
<center><table COLS=2 WIDTH="81%" >
<tr><td>Access Count:</td><td>
<%= (++counter)%>
</td></tr>
<tr><td>Access Time:</td><td>
<%= arrangeCreatedTime(new GregorianCalendar())%></td></tr>
<tr><td>Value:</td><td><%= parmVal%>
</td></tr>
<tr><td></td><td><input type = "submit" value = "Back"></td></tr>
</table></center>
</form>
</body></html>

```

Figure 127. Sample1.jsp

To use this sample:

1. Create Sample1.html and Sample1.jsp.
2. Copy Sample1.html and Sample1.jsp to <DocRoot>.
3. Open <http://<hostname>/Sample1.html> in your browser. You'll see a page similar to Figure 128:

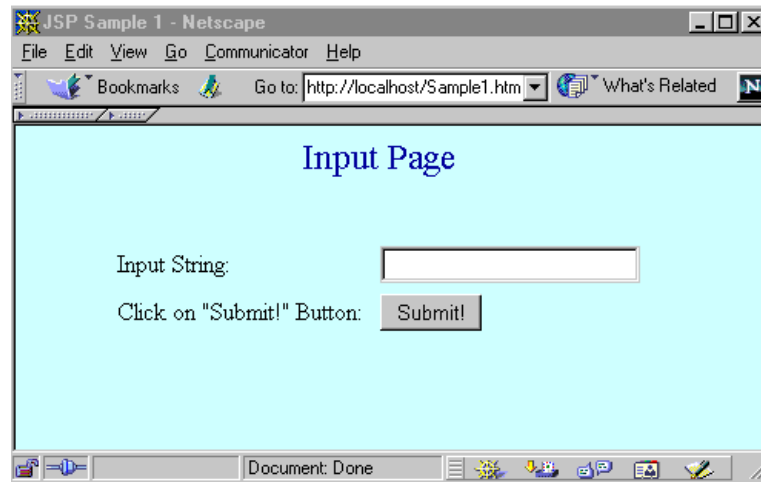


Figure 128. <http://<hostname>/Sample1.html>

4. Update the Input String in the text file and click the **Submit!** button in the following window:

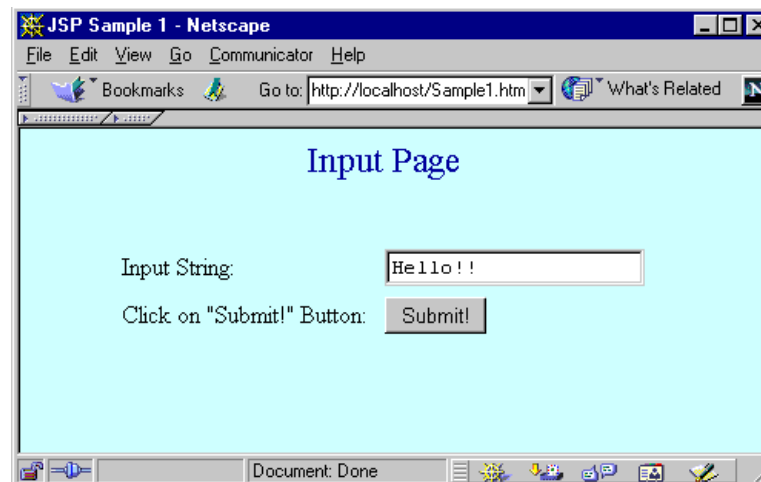


Figure 129. Input String to the Text Field

5. You should receive the result shown in Figure 130 on page 145 from Sample1.jsp.

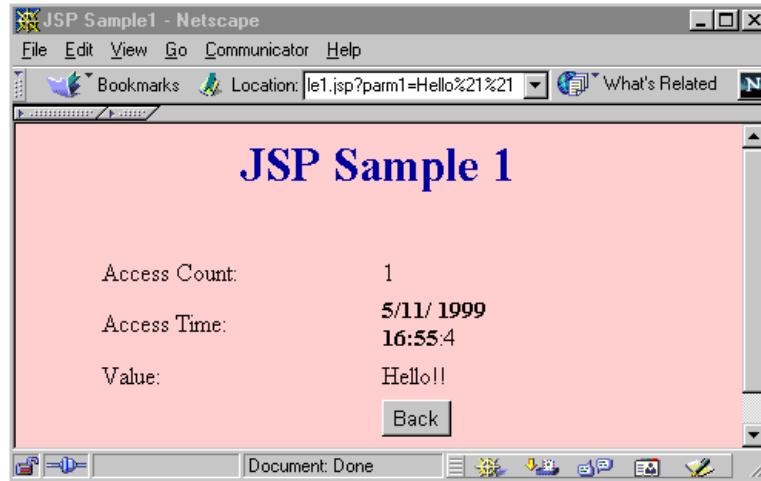


Figure 130. The Result from Sample1.jsp

6. Click the **Back** button on the page. That brings you back to the Input Page.

When you repeat steps 3-6, you get the result faster, since once the servlet instance is created from the JSP, it remains on the JVM unless you unload this instance. Therefore, you get a quicker response, and an incremented Access Count.

This shows you that JSPs give you the same performance advantage as servlets.

3.2.7 JSP Sample 2

Many JSPs are used in the WebSphere samples, for example, WebBank and Xtreme Travel. You can also find HitCount.jsp in <AppServerRoot>/servlet. In this part, we used the sample JSP DisplayData.jsp. This is a simple example of the second assess model for JSPs (Figure 106 on page 123).

3.2.7.1 Sample2 Processing Flow

This is the process flow for Sample2 shown in Figure 131 on page 146.

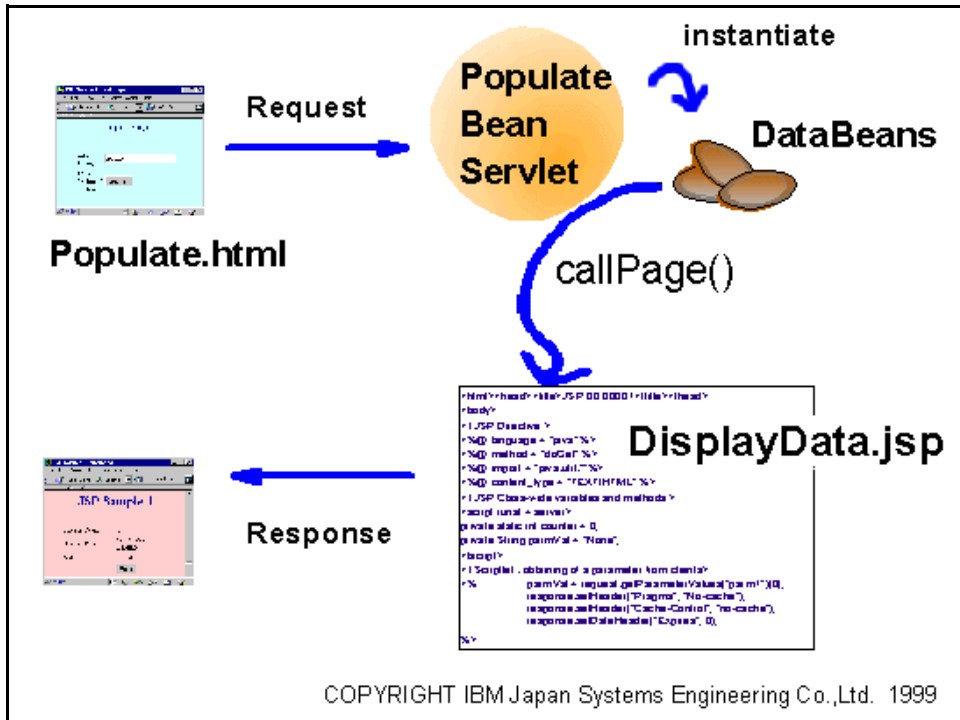


Figure 131. Sample2 Processing Flow

1. A request from Populate.html is sent to PopulateBeanServlet.
2. PopulateBeanServlet creates an instance of the DataBean (named dataBean).
3. PopulateBeanServlet sets the parameters to the dataBean.
4. PopulateBeanServlet sets the Bean instance (dataBean) as an attribute in the current request object.
5. PopulateBeanServlet calls DisplayData.jsp with callPage() method.
6. The client receives the DisplayData.jsp with embedded values of dataBean properties in it.

3.2.7.2 Preparing to Work with the JSP Samples

This sample needs four files:

1. HTML form - We called this Populate.html. It calls PopulateBeanServlet.
2. PopulateBeanServlet.java - This is the servlet that instantiates DataBeans, sets bean values and calls the JSP file (DisplayData.jsp).
3. DataBeans.java - JavaBeans
4. DisplayData.jsp - This gets property values from DataBeans and sends the responses to the user.

With Populate.html, PopulateBeanServlet.java and DisplayData.jsp, you can see sample files at <http://<hostname>/IBMWebAS/doc/whatis/icjssp.html#jsspsamp>.

Click each of the references (Figure 132 on page 147) to see the sample code.

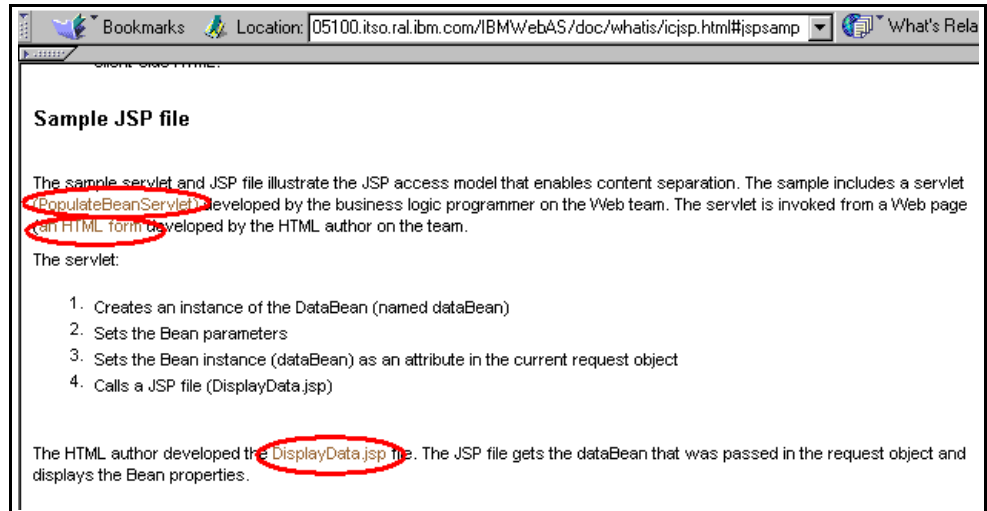


Figure 132. <http://<hostname>/IBMWebAS/doc/whatis/icjsp.html#jpsamp>

Copy and paste from the sample code window, and create these files (Populate.html, PopulateBeanServlet.java, DisplayData.jsp). See Figure 133:

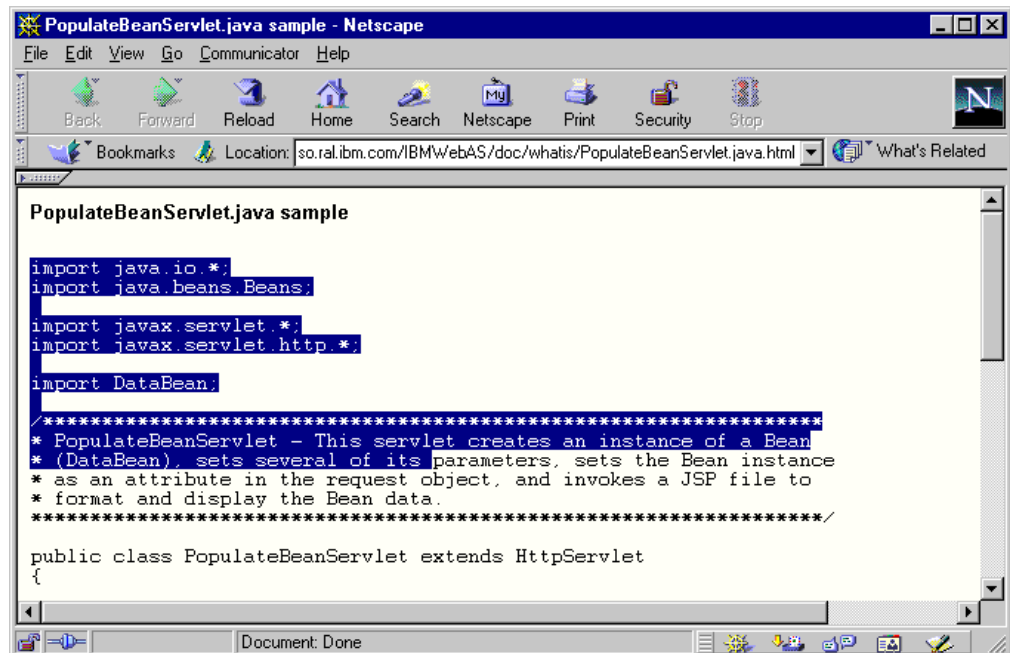


Figure 133. Copy and Paste from the Sample Code Window

```

<!-- This JSP file gets a Bean passed in a request object
      and displays the Bean properties -->

<html>
<head>
<title>Bean Data Display</title>
</head>

<!-- Get the Bean using the BEAN tag -->
<bean name="dataBean" type="DataBean" introspect="no" create="no"
scope="request">
</bean>
<body>
<!-- There are three ways to access Bean properties -->
<!-- Using a JSP scriptlet -->
<% out.println("The value of Bean property 1 is " + dataBeans.getProp1());
%>

<!-- Using a JSP expression -->
<p>The value of Bean property 2 is
<%= dataBean.getProp2() %> </p>

<-- Using the INSERT tag -->
<p>The value of Bean property 3 is
<insert bean=dataBean property=prop3 default="No property value" >
</insert></p>

</body>
</html>

```

Figure 134. DisplayData.jsp

Note: In DisplayData.jsp, there is a spelling error. The error is the bold character in Figure 134 on page 148. You must correct this to be "dataBean", not "dataBeans". If you forget this, you will get an error message from the server (Figure 135) in the next step:

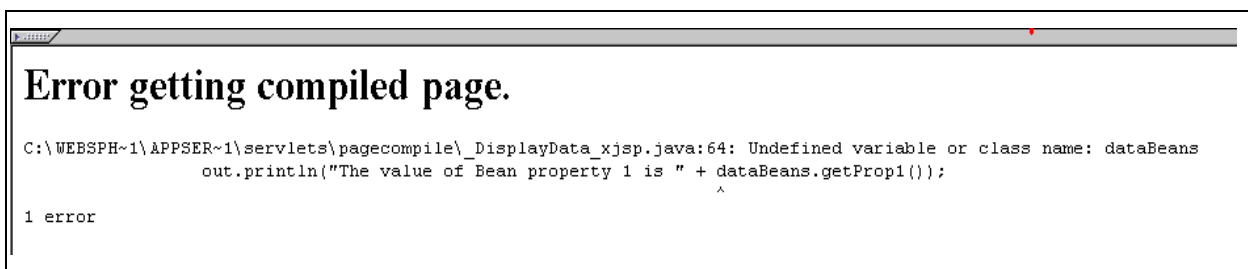


Figure 135. Error Getting Compiled Page

This shows us that something is wrong with the DisplayData.jsp file (Figure 134). Change "dataBeans" to "dataBean", then save the document in <AppServerRoot>/servlets.


```

import java.io.*;
import java.beans.Beans;
import javax.servlet.*;
import javax.servlet.http.*;
import DataBean;

/*****
 * PopulateBeanServlet - This servlet creates an instance of a Bean
 * (DataBean), sets several of its parameters, sets the Bean instance
 * as an attribute in the request object, and invokes a JSP file to
 * format and display the Bean data.
 *****/

public class PopulateBeanServlet extends HttpServlet
{
    public void service(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        DataBean dataBean; // Create an instance of DataBean

        try{
            dataBean = (DataBean) Beans.instantiate(this.getClass().getClassLoader(),
            "DataBean");
        }catch (Exception ex) {
            throw new ServletException("Can't create BEAN of class DataBean: "
            + ex.getMessage());
        }
    }
}

```

Figure 136. *PopulateBeanServlet.java(1/2)*

```

// Set some Bean properties (content generation)
dataBean.setProp1("Value1");
dataBean.setProp2("Value2");
dataBean.setProp3("Value3");

// To send the Bean to a JSP file for content formatting and display
// 1) Set the Bean as an attribute in the current request object
((com.sun.server.http.HttpServiceRequest)
req).setAttribute("dataBean", dataBean);

// 2) Use callPage to invoke the JSP file and pass the current request
object
((com.sun.server.http.HttpServiceResponse)
res).callPage("/DisplayData.jsp", req);
}
} /* end of class PopulateBeanServlet */

```

Figure 137. *PopulateBeanServlet.java (2/2)*

Copy *Populate.html*, *DisplayData.jsp* to your HTTP document root (Figure 138 on page 150). In this case, we used the IBM HTTP Server V1.3.3 on Windows NT. It was located at `\Program Files\IBM HTTP Server\htdocs`.

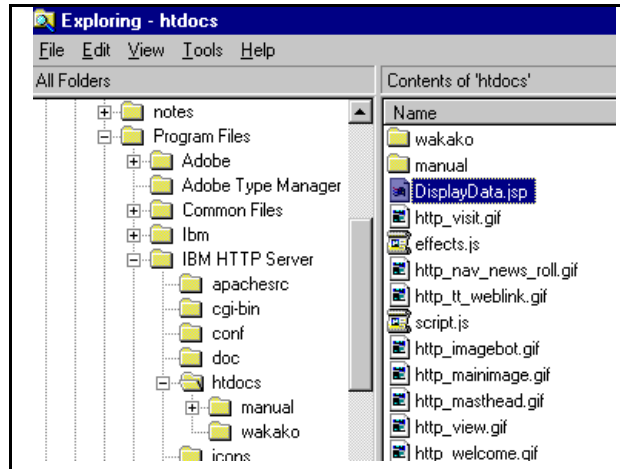


Figure 138. Copy Populate.html, DisplayData.jsp to Your HTTP Document Root

If you look at PopulateBeanServlet.java, you may have the following question: "Where did the **DataBean** come from?". If you compile this file, you receive the following error message:

```
C:\WEBSPH-1\APPSER-1\servlets>javac PopulateBeanServlet.java
PopulateBeanServlet.java:7: Class DataBean not found in import.
import DataBean;
      ^
1 error
```

We couldn't find this package in the WebSphere directories, so we created our own example of DataBean.java (Figure 140 on page 151) for this sample. This was placed in <AppServerRoot>/servlets as PopulateBeanServlet.java.

Note: In PopulateBeanServlet.java, "service" method must be written in lowercase. We found "Service" method was written in this document, so we corrected it.

Now compile DataBean.java and PopulateBeanServlet.java with the following commands:

```
C:\WEBSPH-1\APPSER-1\servlets>javac DataBean.java
C:\WEBSPH-1\APPSER-1\servlets>javac PopulateBeanServlet.java
```

Figure 139. Compile DataBean.java, PopulateBeanServlet.java

```

import javax.servlet.http.*;
import java.io.PrintWriter;
import java.io.IOException;

public class DataBean extends HttpServlet {
    String Prop1 = "DefaultVauleOfProp1";
    String Prop2 = "DefaultVauleOfProp2";
    String Prop3 = "DefaultVauleOfProp3";
    public void service(HttpServletRequest req, HttpServletResponse res)
throws IOException
    {
        PrintWriter writer = res.getWriter();
        writer.println("<BR><B>This is output form DataBean</B><P>");
    }

    public void setProp1(String Prop1) {
this.Prop1 = Prop1;
    }
    public String getProp1() {
return Prop1;
    }
    public void setProp2(String Prop2) {
this.Prop2 = Prop2;
    }
    public String getProp2() {
return Prop2;
    }
    public void setProp3(String Prop3) {
this.Prop3 = Prop3;
    }
    public String getProp3() {
return Prop3;
    }
}

```

Figure 140. *DataBean.java*

3.2.7.3 How to Use This Sample

Open page <http://<hostname>/Populare.html> in your browser. You can see this page in Figure 141:



Figure 141. <http://<hostname>/Populate.html>

```
<HTML>
<head><title>Run PopulateBeanServlet</title></head>
<BODY>
<H1>Run PopulateBeanServlet</H1>
<P>Do you want to run the PopulateBeanServlet?
<FORM action="/servlet/PopulateBeanServlet" method="GET">
<INPUT type="SUBMIT" value="Yes">
<INPUT type="SUBMIT" value="No">
</FORM>
</BODY>
</HTML>
```

Figure 142. Populate.html

Click the **Yes** or **No** button to call PopulateBeanServlet.

You should receive the result shown in Figure 143:

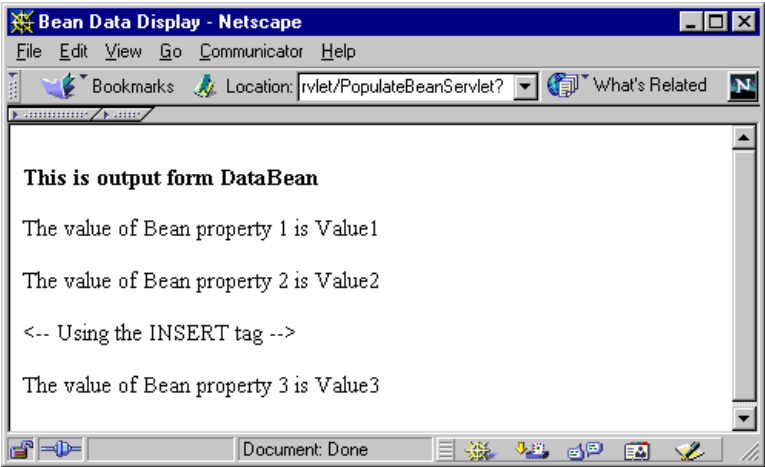


Figure 143. Result of JSP Sample

You receive this result whether you click **Yes** or **No**. The developer might have forgotten to code this part, or left it to the reader as an exercise to complete.

3.2.8 Tools for Creating JSP Files

You can use various tools to make JSP. Please refer to 2.6, "Setting Up a Development System" on page 86.

3.3 Using the WebSphere XML Tools

eXtensible Markup Language (XML) provides a way of formatting data with tags so that it is platform and application independent. It is derived from SGML but XML attempts to alleviate some of HTML's shortcomings by promoting the separation of presentation and content and building in extensibility. There is a good introduction to XML in the WebSphere documentation at <http://<ASRoot>:9527/doc/whatis/icxml14j.html>. There is also more information at <http://www.ibm.com/xml> and <http://www.alphaworks.ibm.com>.

XML allows the application designer to describe data in a way that represents the structure and internal semantics of that data. It is then possible to take the XML and transform it so that it can be viewed by a client, either as HTML or directly, as formatted XML, through the use of a style sheet. WebSphere provides XML document structure services to facilitate the creation and manipulation of XML documents and allow the development of XML-based applications. XML documents can be static or created dynamically through the use of servlets, JSPs or EJBs accessing various back-end data sources such as databases. See 3.3.6, “Example: Using XSL and XML to Format DB2 Data” on page 166 for an example of a servlet that creates dynamic XML from a database.

In this section we look at the document structure services provided with WebSphere and then at how to extend these services to produce formatted XML data from a database. To generate and process the XML, we use the standard XML document structure services and also additional tools from the IBM alphaWorks Web site at:

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

In addition, there is a discussion group available at alphaWorks for LotusXSL and XML4J.

3.3.1 Environment

This section describes the environment that we used to produce the information in this chapter.

3.3.1.1 Hardware and Software Environment

The following table lists the hardware and software environment used in this section:

Table 10. XML Hardware and Software Environment

Item	Version
Machine	Netfinity 3000 PII 350MHz, 128 MB RAM
Operating System	Windows NT V4.0 with service pack 4
Application Server	WebSphere Application Server 2.02 Advanced
Database	DB2 UDB V5.2 Workgroup Edition installed from the WebSphere 2.0 Advanced CD

3.3.1.2 External Tools Used

Table 11 is a list of the tools that we used in addition to the tools installed with WebSphere. To get to the individual Web page for each tool do the following:

- Go to <http://www.alphaworks.ibm.com>.
- Select **Technologies**.
- Scroll to the bottom of the yellow list on the left of the page and select **XML**.

- Select the technology in the yellow list.

Table 11. *alphaWorks Technologies Used*

Tool	Version	Version shipped with WebSphere V2.02	Description
Lotus XSL	0.16.5	0.16.2	XML style sheet processor for transforming XML to other formats such as HTML.
XML Parser for Java	2.06	1.1.14	Update to WebSphere tool used to parse XML files for processing by Java programs.

Note: XML4J V2.x is much faster than V1.x.

3.3.2 Setting Up the Environment

To work with the XML Document Services you need to set up the environment correctly. This involves adding the IBM XML for Java parser JAR files to the application server classpath. To do this follow these steps:

1. Open the WebSphere administration application by loading `http://<your server name>:9527/` and log in with your administration user ID and password.
2. Expand the **Setup** category and select **Java Engine**.
3. In the Application Server classpath field add the following directory to the end of the classpath (classpath entries are separated by semicolons on Windows and colons on AIX):

```
<server root>\lib\xml4j.jar;
```

Note: On Windows, any directory path must include directories in the 8.3 file naming format. You can see these name by typing `dir /x` instead of `dir` to get a directory listing. Typically, directories with names longer than eight characters will have a tilde (~) character included followed by a number. For example the directory path above may be:

```
d:\WEBSPH-1\APPSER-1\lib\xml4j.jar;
```

on Windows.

4. Stop and restart your Web server to make the changes active.

Note: While working with the XML document structure services on Windows NT (using XML4J on Win32 Symantec JIT 1.1.6), we had problems with some versions of the XML parser that had the JIT compiler turned on. See 3.3.6.10, “Description of Errors Encountered during Development” on page 181 for a full description of the symptoms and their resolution.

3.3.3 Processing XML

To make use of XML information it is necessary to transform it from a text format into a data structure in memory that a program can use. This process of reading the source and making sense of it is called parsing. To parse XML, a parser that can understand XML semantics is required. WebSphere provides a version of the IBM XML for Java parser (also known as XML4J) as part of XML document structure services. It should be noted that there are later versions of the parser

available on the alphaWorks Web site. See 3.3.7, “Installing Later Versions of the XML Tools” on page 185 for instructions on how to configure WebSphere to use the upgraded parser. In addition, XML4J and LotusXSL components of XML Document Structure Services are following an open source development model. The object is shipped and supported in WebSphere but new releases will be made available via the alphaWorks Web site.

The component XML4J provides two different methods for processing XML:

1. Event-driven parsing using the Simple API for XML or SAX parsing

Code should be portable between XML processors. They are only interested in processing pieces of the overall document (or if you have a very large document). If you are using 2.x-level classes it provides the fastest processing) and is easier to use than DOM tree parsing.

2. Event-driven parsing using event handlers

Event handlers are easier to use than SAX. This is because you can indicate specific elements or attributes that you care about.

3. DOM Tree Parsing

This is the most portable of the methods for processing XML. Microsoft implements this in Internet Explorer V5. It lets you process the entire document tree in memory to manipulate the document structure. It is slower to parse than the other methods and it uses more resources.

In event-driven parsing the application using the parser registers with the parser a set of events in which it is interested. As the parser processes the XML it recognizes data within the XML that makes up logical units called tokens. As each token is recognized the parser checks to see if any applications have registered their interest in the particular token encountered and signals appropriate events to those applications.

In tree parsing the parser constructs a tree data structure in memory corresponding to the logical structure of the input XML. Each logical unit or token is placed at a node of the tree and tokens logically contained within other tokens are placed as children of the containing token. The data structure is then made available to the application to process as needed.

For example the XML source fragment shown in Figure 144 might produce the logical tree structure shown in Figure 145 (the actual structure would be significantly more complex than this. See 3.3.3.3, “Parsing Using the DOM Tree Parser” on page 160 for further details).

```
<person>
  <name>
    <first>John</first>
    <last>smith</last>
  </name>
  <age>34</age>
</person>
```

Figure 144. Sample XML Fragment

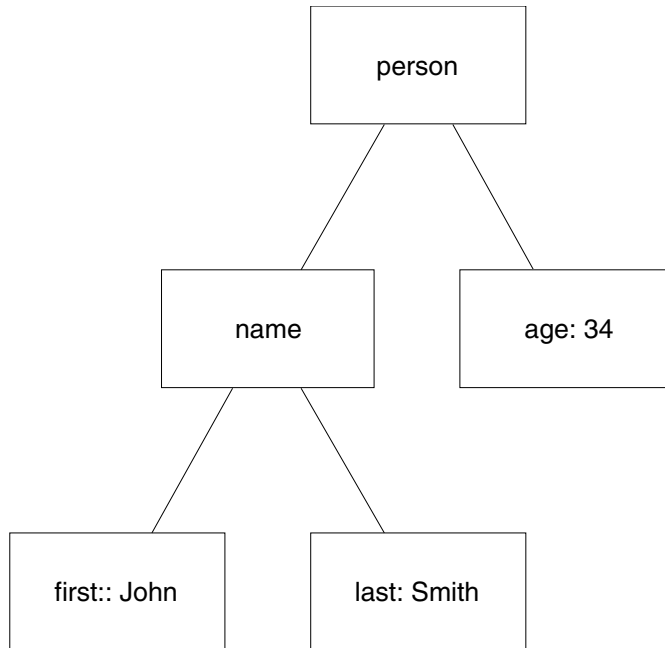


Figure 145. Abstract Representation of the DOM Structure

One advantage of event parsing over tree parsing is that the whole XML document structure does not have to be kept in memory at once, allowing very large documents to be processed. Another advantage is that applications need only register themselves for events in which they are interested. Applications using trees generated using a tree parser must deal with every element in the tree structure, whether or not it is significant to them. The advantage of tree parsers over event parsers is that modifications to the tree structure are relatively easy to make allowing transformations to be made. This is useful when the data must be manipulated to produce a different output form such as in XSL processing (see 3.3.5, “XML Style Sheets and LotusXSL” on page 164).

The choice of which processing method to use is driven by the needs of your application.

3.3.3.1 SAX Event-Driven Parsing

In SAX parsing, the parser produces a simple set of events for the application at the document level. The application must register a handler for these events with the parser. Typically, the handler will need to implement the interface `org.xml.sax.DocumentHandler`. This is the basic interface for SAX parsing. Handlers wishing to register for SAX document events must implement each of the methods shown in Table 12, although the method implementations need not do anything. See file:///server/root/web/doc/apidocs/org.xml.sax.DocumentHandler.html#_top_ for more details.

Table 12. Methods in the `org.xml.sax.DocumentHandler` Interface

Method	Purpose
<code>characters(char[], int, int)</code>	Receive notification of character data.

Method	Purpose
endDocument()	Receive notification of the end of a document.
endElement(String)	Receive notification of the end of an element.
ignorableWhitespace(char[], int, int)	Receive notification of ignorable white space in element content.
processingInstruction(String, String)	Receive notification of a processing instruction.
setDocumentLocator(Locator)	Receive an object for locating the origin of SAX document events.
startDocument()	Receive notification of the beginning of a document.
startElement(String, AttributeList)	Receive notification of the beginning of an element.

To implement SAX event-driven parsing, the application must do the following (more details can be found in the WebSphere documentation at <ASRoot>\web\doc\howto\itxml4j.html#evtparse):

1. Put the following two lines with the import statements at the top of the java source file to include the java classes:

```
import org.xml.sax.*;
import com.ibm.xml.parser.SAXDriver;
```

2. Implement the org.xml.sax.DocumentHandler interface:

```
public class parseXMLUsingSAX extends HttpServlet implements DocumentHandler
```

3. Create the parser:

```
Class parserClass = Class.forName("com.ibm.xml.parser.SAXDriver");
Parser parser = (Parser) parserClass.newInstance();
```

4. Register the class with the parser as a document event listener:

```
parser.setDocumentHandler(this);
```

5. Run the parser:

```
parser.parse(new InputSource(bais));
```

Where bais is a java.io.ByteArrayInputStream. You could use any sort of java.io.InputStream here.

The parser will now look at the input stream and call the event methods that were defined in the class.

Example source code used by the Xtreme XML sample to perform SAX parsing can be found at <ASRoot>\samples\XtremeXML\parseXMLUsingSAX.java.

For more information on the SAX standard visit the official Web site

<http://www.microstar.com/sax.html> or the site of one of the creators of SAX,

<http://www.megginson.com/SAX/>.

3.3.3.2 Parsing Using Element Handlers

The second type of parsing that XML document structure services supports is using element handlers. The parser that this method uses is the same as the DOM tree parser (see 3.3.3.3, “Parsing Using the DOM Tree Parser” on page 160), but this method of parsing uses events. Thus, this method of parsing is a hybrid between true event-driven parsing, such as that provided by the SAX parser, and DOM tree parsing. Parsing using element handlers involves registering handlers for different elements within the document as the parser creates the DOM tree. These handlers are called whenever the parser encounters an element of the specified type. It is also possible to register a general handler to receive events when any element is encountered.

Parsing using element handlers is most useful when you need to make changes to the structure of a DOM tree as it is being constructed. Each element that needs to be modified has a handler registered to it. When the parser encounters an element of that type it passes the document element to the handler which allows the handler to effect the transformation.

The disadvantage of this method is that it needs to construct the entire DOM tree in memory in order to generate the events. The advantage is that handlers can be registered selectively for only the document elements that you are interested in which leads to increased code efficiency.

To process XML using element handlers you need to create an element handler and then register it with the parser. To create an element handler do the following:

1. Add the following import lines to the top of your element handler code:

```
import com.ibm.xml.parser.*;
import org.w3c.dom.*;
```

2. Implement the `com.ibm.xml.parser.ElementHandler` interface (see file:///<ASRoot>/web/doc/apidocs/com.ibm.xml.parser.ElementHandler.html#_top_ for API details). For example:

```
public class itineraryHandler implements ElementHandler
```

3. Create an implementation of the `handleElement` method that has as a parameter a `com.ibm.xml.parser.TXElement` object. This is the element that the parser created and for which your handler has been registered. After performing processing using the DOM APIs (see file:///<ASRoot>/web/doc/apidocs/com.ibm.xml.parser.TXElement.html#_top_), the method must return either a modified `TXElement` object or null if the element is to be removed from the DOM tree.

An example element handler can be found in `<ASRoot>\samples\XtremeXML\itineraryHandler.java`.

To create a Java application that can register and use the element handler with the parser, perform the following steps:

1. Import the required packages by placing the following lines at the top of your code:

```
import com.ibm.xml.parser.*;
import org.w3c.dom.*;
```

2. Import the package or class containing the element handler or handlers that you wish to use. For example:

```
import itineraryHandler;
```

3. Create a parser instance to work with, specifying a file to use for error messages:

```
Parser parser = new Parser("xslparse.err");
```

4. Register the element handler or handlers created with the parser by calling the `parser.addElementHandler` method. For example, to add a handler for the *outbound-airline* element the following code could be used:

```
parser.addElementHandler(new itineraryHandler(), "outbound-airline");
```

If a general element handler is to be registered, that is, one that is called for all elements rather than a specific element, then do not include the second string parameter with the element name.

5. Call one of the `readStream` methods on the parser to read the XML input stream and call the handlers. When the `readStream` method is called, the DOM tree is built and the handlers that are registered for each element are called in order of their registration, with the handlers for specific tags being called before any general handlers. For example, to read and process the XML stored in a `java.io.ByteArrayInputStream` variable (`bais`), you could code the following:

```
doc = parser.readStream(bais);
```

An example of a program that uses a handler to perform transformations in the DOM tree prior to processing can be found in `<ASRoot>\samples\XtremeXML\parseXMLUsingElementHandlers.java`.

It is also possible to register a number of other handlers to work with different sorts of events, other than element events. Table 13 shows the different methods that can be called on a `com.ibm.xml.parser.Parser` object to register different sorts of handlers with the parser. It also shows the circumstances under which each element handler's methods are called and the name of the interface that must be implemented for each different sort of handler. Full details about the API for these handlers can be found at file:///web/doc/apidocs/com.ibm.xml.parser.Parser.html#_top_.

Table 13. Register Event Handlers for `com.ibm.xml.parser.Parser` Instances

Method to Register	When Handler is Called	Handler Interface Name (in <code>com.ibm.xml.parser</code>)
<code>addElementHandler</code>	Called when an element tag is recognized. If an optional element name is specified it will be called for only that element; otherwise, the handler will be called for all elements.	<code>ElementHandler</code>

Method to Register	When Handler is Called	Handler Interface Name (in com.ibm.xml parser)
addNoRequiredAttributeHandler	Called when the parser detects that an attribute that is marked as required on the DTD has not been included on an element.	NoRequiredAttributeHandler
addPIHandler	Called when the parser recognizes an XML processing instruction.	PIHandler
addPreRootHandler	Called after the DTDs (if any) have been parsed but before any elements in the XML document have been parsed.	PreRootHandler
setTagHandler	Called when an element <i>tag start</i> and <i>tag end</i> are recognized.	TagHandler
setReferenceHandler	Called when a general entity reference is recognized (prior to parsing any entity streams) and after all entity streams for a general reference have been parsed.	ReferenceHandler

3.3.3.3 Parsing Using the DOM Tree Parser

DOM tree parsing is the most sophisticated, and hence the most complex, method of parsing XML. Unlike event-driven parsing, such as SAX parsing and parsing using element handlers, DOM tree parsing is performed after the parser has done its work and constructed a data structure called a Document Object Model tree or DOM tree. This tree is then navigated by various methods and can also be manipulated.

The abstract specification for the DOM was created by the World Wide Web Consortium (known as W3C) and is available online at <http://www.w3.org/TR/PR-DOM-Level-1/>. W3C also specified Java language bindings for the DOM in the package org.w3c.dom. These bindings can be found in an appendix to the DOM specification at <http://www.w3.org/TR/PR-DOM-Level-1/java-language-binding.html>. WebSphere includes javadoc documentation for the org.w3c.dom package at `file:///<ASRoot>/web/doc/apidocs/Package-org.w3c.dom.html`. WebSphere also includes an IBM implementation of the DOM Java language bindings in the package com.ibm.xml.parser, and the complete API details are available at `file:///<ASRoot>/web/doc/apidocs/Package-com.ibm.xml.parser.html`.

Working with DOM parsing involves two steps: first creating the DOM tree from the input source, and secondly navigating the tree to process the XML. The following steps are required to produce a DOM tree in an application:

1. Add lines to the code to import the W3C and IBM DOM packages such as the following:

```
import com.ibm.xml.parser.*;
import org.w3c.dom.*;
```

2. Create an instance of the parser specifying the location of the error file:

```
Parser parser = new Parser("xslparse.err");
```

3. Use one of the readStream methods to parse the input XML and return a reference to the root of the parsed XML tree. For example, the following line of code reads input XML from a java.io.ByteArrayInputStream(bais) and stores it in a com.ibm.xml.parser.TXDocument object doc:

```
doc = parser.readStream(bais);
```

Once you have a reference to the root of the document tree you can then begin to process the elements within that tree. Table 14 on page 161 shows the types of entities that you may encounter in processing a DOM tree along with a brief description and a reference to the package in which they reside. For further detail, the API documentation is installed with WebSphere at file:///<ASRoot>/web/doc/apidocs/packages.html and the DOM documentation can be found at <http://www.w3.org/TR/PR-DOM-Level-1/>.

Table 14. Data Item Types Found in a DOM Tree

Data Item	Description	Implementation Class	Implementation Package
Document	The whole document tree. The Document object contains all the other objects.	TXDocument	com.ibm.xml.parser
DocumentFragment	A part of a document. Document fragments do not exist in the tree; they are created and used during processing.	TXDocumentFragment	com.ibm.xml.parser
DocumentType	A data structure representing the DTD for this XML document. This may be null if no DTD was specified.	DTD	com.ibm.xml.parser
EntityReference	A reference to the value of a previously defined entity object. Usually the parser replaces the EntityReference with the value of the referred entity object.	EntityReference	org.w3c.dom
Node	Node is the parent interface for all of the objects in the DOM tree. Methods on the node element are the primary means of navigation about the tree.	Node	org.w3c.dom

Data Item	Description	Implementation Class	Implementation Package
Element	An element in a document. Most of the objects in a document will be Elements.	TXElement	com.ibm.xml.parser
Attribute	An attribute associated with an Entity. Attributes will only be children of the owning Entity node.	TXAttribute	com.ibm.xml.parser
ProcessingInstruction	An instruction to an XML processor.	TXPI	com.ibm.xml.parser
Comment	A comment placed in the XML.	TXComment	com.ibm.xml.parser
Text	Content inside the start and end tags of an element that is not itself an element definition.	TXText	com.ibm.xml.parser
CDATASection	Section of text in which data with tags may be included without interpretation by the parser.	TXCDATASection	com.ibm.xml.parser
Entity	Entities are used to provide a shorthand for something else such as special characters or binary data.	EntityDecl	com.ibm.xml.parser
Notation	Notations are a way of telling external applications the format of the data preceding them. For more details see the XML 1.0 spec, section 4.7.	TXNotation	com.ibm.xml.parser

In practice, most of the work that is done in processing DOM trees uses the Node, Document and Element objects. The following are a few common techniques used in navigating and processing a DOM tree:

- Use the `hasChildren()` method on the Node interface to determine if a Node has children.
- Use the `getFirstChild()`, `getLastChild()`, `getNextSibling()`, `getPreviousSibling()` and `getParentNode()` methods on the Node interface to navigate around the tree.
- Use the `appendChild(Node)`, `insertBefore(Node,Node)`, `removeChild(Node)` and `replaceChild(Node)` on the Node interface to manipulate the children of a node.
- Use the `getNodeTypes()` method on the Node interface to determine a Node's type.
- Use the `createXXXX` methods on the Document interface to create new objects for insertion into the DOM tree.
- Use the different `getAttribute`, `setAttribute` and `removeAttribute` methods on the Element interface to manipulate Attributes.
- Use the `searchXXXX` and `getNthElementXXXX` methods on the Element interface to find different descendants.

The parseXML example found in <ASRoot>\samples\XtremeXML\parseXML.java shows the use of a number of these techniques.

3.3.4 XML Catalogs

WebSphere includes a catalog of public XML DTDs for use by applications. Catalogs are useful when you have valid XML or when you are using industry standard DTDs, and you want to abstract the exact locations of the DTDs. Figure 147 on page 164 shows a short Java program that gives an example of how to use the catalog feature to provide a DTD for a small XML sample OTP_v_0-99.xml as shown in Figure 146 on page 163.

```
<?xml version="1.0" ?>
<!DOCTYPE OtpMessage PUBLIC "-//Open Trading Protocol//DTD Open Trading
Protocol//EN" "OTP_v_0-99.dtd" >
<OtpMessage>
  <TransRefBlk ID="M1">
    <TransId ID="M1.1" OtpTransId="A1" OtpTransType="A2"
TransTimeStamp="A3"/>
    <MsgId ID="M1.2" xml:lang="us-en" SoftwareId="B2"/>
  </TransRefBlk>
  <PingReqBlk ID="M2"/>
</OtpMessage>
```

Figure 146. OTP_v_0-99.xml

```

package com.ibm.pjk.xml;

import com.ibm.xml.parser.*;
import java.io.*;
/**
 * This type was created in VisualAge.
 */
public class CatalogTester {
/**
 * CatalogTester constructor comment.
 */
public CatalogTester() {
super();
}
/**
 * This method was created in VisualAge.
 * @param args java.lang.String[]
 */
public static void main(String args[]) {
    TXDocument doc = null;
    String fname = "d:\\data\\catalog\\xmlparse.err";
    String catalogFilespec =
"d:\\WebSphere\\AppServer\\web\\xml\\grammar\\dtd\\dtd.cat";
    String xmlSource = "d:\\data\\catalog\\OTP_v_0-99.xml";
    Stderr se = new Stderr(xmlSource);
    try {
        FileReader rr = new FileReader(catalogFilespec);
        se.loadCatalog(rr);
        rr.close();
    } catch (java.io.IOException e) {
        e.printStackTrace();
    }
    Source src = null;
    try {
        src = se.getInputStream(fname, null,
Stderr.file2URL(fname).toString());
    } catch (java.io.IOException e) {
        e.printStackTrace();
    }
    Parser pc = new Parser(xmlSource, se, se);
    doc = pc.readStream(src);
}
}

```

Figure 147. Sample Java Code to Use an XML Catalog

This XML sample uses a public DTD for the Open Trading Protocol. More details on the OTP are available at <http://www.otp.org/>.

3.3.5 XML Style Sheets and LotusXSL

To present XML data on a client machine it has to be formatted graphically in some way. HTML handles this problem by defining a fixed set of tags that define content and formatting at the same time. XML, on the other hand, seeks to separate the content of the data from how it is presented. The eXtensible Style sheet Language (XSL) seeks to complete the picture by defining a set of

formatting elements that can be added to XML documents to describe how they should be presented to a client. More importantly, it is a mechanism for transforming an XML document to add those formatting elements without affecting the source XML.

The XSL standard, which at the time of writing had not been finalized, consists of two parts:

- A language for transforming XML documents

This is a way for a style sheet to specify which elements in an XML document are to be transformed and how they are to be transformed. The idea is that a parsed XML source DOM tree is input to the XSL processor and the tree is modified to produce an output DOM tree based on instructions placed in an XSL style sheet. This mechanism, also called XSL transformations (XSLT), was originally designed to allow the addition of formatting tags to an XML document. It is also useful in a more general sense for specifying other transformations to XML documents, such as formatting them into HTML. More information on XSLT can be found at <http://www.w3c.org>. The latest draft of the specification available at the time of writing was at <http://www.w3.org/TR/1999/WD-xslt-19990421>.

- An XML vocabulary for specifying formatting semantics

For an XSL formatter to correctly present a graphical view of an XML document to a client there must be a well-understood set of formatting commands for the formatter to process in constructing the output. The XSL formatting language defines a rich set of formatting tags using an XML DTD to define the formatting language and a specification to define the meanings of the tags that comprise the formatting language. These tags can then be interpreted and processed by an XSL formatter. This language is described further in the documentation at <http://www.w3c.org>. The latest draft of the specification available at the time of writing may be found at <http://www.w3.org/TR/WD-xsl/>.

Another good source of information on XSL is the online magazine XML.com (<http://www.xml.com>) which has a number of XSL articles at <http://www.xml.com/xml/pub/Style>. Articles here need to be read carefully as the XSL specification may have been different when the article was written. The documentation for the tool being used is always the final arbiter.

To understand XSL fully it is also useful to understand the semantics of XML namespaces. One good article to look at would be "XML Namespaces by Example" by Tim Bray at <http://www.xml.com/xml/pub/1999/01/namespaces.html>. The W3C specification for namespaces can be found at http://www.w3.org/TR/REC-xml_names/.

To facilitate XSL transformations, WebSphere XML Document Structure Services includes a version of the LotusXSL tool which can be used to effect XSL transformations using XSL style sheets. It can be found in the lib directory (<ASRoot>/lib/lotusxsl.jar). At the time that WebSphere 2.02 was built the XSL standard was still very immature, but it has undergone considerable change. It is suggested that a newer version of LotusXSL from <http://www.alphaworks.ibm.com> be installed to process XSL (see 3.3.7, "Installing Later Versions of the XML Tools" on page 185 for information on how to do this). At the time of writing, the latest version available was 0.17.1.

3.3.6 Example: Using XSL and XML to Format DB2 Data

To illustrate the uses of XSL we created a sample servlet using LotusXSL to process XML data generated from a DB2 query so that it could be output to a browser as HTML. The following sections will show you the relevant features of the servlet and explain the relevant features of XSL. References in the text will point to more complete reference documentation.

3.3.6.1 Setting Up the Environment

To work with LotusXSL we installed a later version of the processor from the alphaWorks Web site and a later version of XML4J. It is recommended that before working with XSL you download and install the latest versions, since the XSL standard is continually evolving. See 3.3.7, "Installing Later Versions of the XML Tools" on page 185 for details on how to perform this task. We installed LotusXSL Version 0.16.5 and XML4J Version 2.0.6. We also tried the code with LotusXSL Version 0.17.0, which became available during the writing of this section (see 3.3.6.11, "Release 0.17.0 of LotusXSL" on page 184).

Already present on the machine were DB2 and WebSphere Application Server 2.02 Advanced Edition. The DB2 sample database had been created.

3.3.6.2 Creating a DTD

The first thing we did in our XML project was to create a Data Type Definition (DTD) describing the data that we wanted to model. You only need a DTD if you want to validate the XML data against a schema. Figure 148 on page 167 shows the DTD in its finished form. The data we wanted to model was the employee table from the DB2 sample database. Note the following points:

- Each row in the employee table represents a single employee, but we want to model a number of employees. Hence the top level element of our document is called "employees". We define the top level element thus:

```
<!ELEMENT employees (employee)+>
```

meaning that an "employees" consists of one or more "employee" elements as denoted by the "+" sign.

- The employee element is declared as having name, phone, hiredate, title, birthdate, salary, bonus and commission components and empnum, edlevel, department and sex attributes. This division between attributes and child elements is somewhat arbitrary, but the usual rule of thumb is that if the element has an existence or structure of its own apart from its owning element, then it should be an element; otherwise it should be an attribute. It really doesn't matter in the long run and a different division could have been argued. The child components with structure, namely name, birthdate and hiredate definitely need to be elements, since they have attributes of their own.
- The element "name" is declared as empty because all of its data is contained in its attributes, first, midinit and last, and so there is no need to specify the content of this element. We will see later that this means that the "name" element is its own start and end tag. The "name" element brings together the first name, middle initial, and last name columns in the employee table into a single element, giving the flexibility to treat the name as a whole while still allowing access to its individual components.

- The "hiredate" and "birthdate" elements, while being empty elements like the "name" element, take the opposite approach in terms of combining columns, in that they take single columns in the database and split them so that their individual components, the day, month, and year, are accessible. We will see later that this allows great flexibility in formatting the dates in XML (see 3.3.6.4, "Developing an XSL Style Sheet" on page 169 for examples of the different ways that dates were formatted).

```

<?xml encoding="US-ASCII"?>
<!-- Revision: 1.0, xml4j2, xml4j2_0_6 -->
<!-- employee.dtd -->
<!ELEMENT employees (employee)+>
<!ELEMENT employee (name,
                    phone,
                    hiredate,
                    title,
                    birthdate,
                    salary,
                    bonus,
                    commission)>
<!ATTLIST employee empnum ID #REQUIRED
                  department CDATA #REQUIRED
                  edlevel CDATA #REQUIRED
                  sex CDATA #REQUIRED>
<!ELEMENT name EMPTY>
<!ATTLIST name first CDATA #REQUIRED
            midinit CDATA #IMPLIED
            last CDATA #REQUIRED>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT midinit (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT hiredate EMPTY>
<!ATTLIST hiredate year CDATA #REQUIRED
                month (01|02|03|04|05|06|07|08|09|10|11|12) #REQUIRED
                day
                (01|02|03|04|05|06|07|08|09|10|11|12|13|14|15|16|17|18|19|20|21|22|23|24|25|26|27|28|29|30|31) #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT birthdate EMPTY>
<!ATTLIST birthdate year CDATA #REQUIRED
                month (01|02|03|04|05|06|07|08|09|10|11|12) #REQUIRED
                day
                (01|02|03|04|05|06|07|08|09|10|11|12|13|14|15|16|17|18|19|20|21|22|23|24|25|26|27|28|29|30|31) #REQUIRED>
<!ELEMENT salary (#PCDATA)>
<!ELEMENT bonus (#PCDATA)>
<!ELEMENT commission (#PCDATA)>

```

Figure 148. The Data Type Definition for the Employee Table

- The day and month attributes for the hiredate and birthdate elements utilize the "|" (OR) operator to define a list of possible values. This makes error detection easier, since wrong values will be detected by the parser and error messages generated. This was found to be very useful during development when it was found that the Java GregorianCalendar class returned month

values in the range 0 to 11 rather than 1 to 12, which caused the parser to complain that "00" was not a valid month and caused some code rewriting to perform the conversion.

The year attribute would have been much more difficult to treat in this manner, and in this case it was left as (#PCDATA). The code that is shown later makes sure that there are no Y2K issues with the data by filtering all values through the Java `GregorianCalendar` class, but to be really safe the DTD could be modified to reject invalid year values. This is left as an exercise for the reader.

- The other elements are declared as (#PCDATA), which means that they contain character data in an unspecified format.

There is an excellent tutorial on the IBM XML developer site describing how to write XML and create DTDs. See

<http://www.software.ibm.com/developer/library/tutorial-prog/writing.html#dtlds> for a more exhaustive treatment of the syntax.

3.3.6.3 Building the First XML Document

In building an XML application that produces dynamically generated XML, it is always useful to have a sample of the finished product to which you can refer. Figure 149 on page 169 shows the XML file that was hand generated using the DB2 tools to look at the data in the database and convert it to XML based on the DTD already developed. You may want to refer back to Figure 148 on page 167 to check the generated data against the DTD.

Ignore for the moment the style sheet processing directive on line 2, but do note the DOCTYPE declaration on line 4 that points back to the DTD and references the top level element `employees`. At the moment this declaration points to a file on the local file system, but when we generate this XML dynamically in 3.3.6.6, "The Servlet Code" on page 175 we will replace this file reference with a URL reference.

```

<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="\data\employee\employee.xsl" ?>

<!DOCTYPE employees SYSTEM "file:/d:/data/employee/employee.dtd">
<employees>
  <employee empnum="E000010"
    department="A00"
    edlevel="18"
    sex="F">
    <name first="SHILI" midinit="I" last="HAAS"/>
    <phone>3978</phone>
    <hiredate year="1965" month="01" day="01"/>
    <title>PRES</title>
    <birthdate year="1933" month="08" day="24"/>
    <salary>52750.00</salary>
    <bonus>1000.00</bonus>
    <commission>4220.00</commission>
  </employee>
  <employee empnum="E000020"
    department="B01"
    edlevel="18"
    sex="F">
    <name first="MICHAEL" midinit="L" last="THOMPSON"/>
    <phone>3476</phone>
    <hiredate year="1973" month="10" day="10"/>
    <title>MANAGER</title>
    <birthdate year="1948" month="02" day="02"/>
    <salary>41250.00</salary>
    <bonus>800.00</bonus>
    <commission>3300.00</commission>
  </employee>
  <employee empnum="E000030"
    department="C01"
    edlevel="18"
    sex="F">
    <name first="SALLY" midinit="A" last="KWAN"/>
    <phone>4738</phone>
    <hiredate year="1975" month="04" day="05"/>
    <title>MANAGER</title>
    <birthdate year="1941" month="05" day="11"/>
    <salary>38250.00</salary>
    <bonus>800.00</bonus>
    <commission>3060.00</commission>
  </employee>
</employees>

```

Figure 149. The Hand-Generated XML File Used in Development

3.3.6.4 Developing an XSL Style Sheet

The next stage in the process is to develop an XSL style sheet to effect the transformations required in the XML data. In this project we developed three style sheets to demonstrate how the same data can be transformed in different ways by using different style sheets.

The method that we used to develop the style sheets was to look at examples, particularly those shipped with the LotusXSL tool, and adapt them to our purposes. The style sheets take the XML data and use pattern templates to recognize tag structures within the XML. These templates then specify what data is to replace the specified element or attribute.

For an introduction to the development of XSL style sheets, see the article by Norman Walsh at <http://www.xml.com/xml/pub/1999/01/walsh3.html> (but note that the XSL semantics have changed slightly since this article was published).

Figure 150 on page 171 shows the first of the style sheets that we developed: `phonelist.xsl`. The following points are worth noting:

- The first step is to declare three namespaces. The first one serves to tell the XSL processor where to find the definitions for the XSL transformation language. We use the older <http://www.w3.org/TR/WD-xsl> definition of the XSL namespace that is compatible with LotusXSL Version 0.16.5. If we were using LotusXSL Version 0.17.0 this value would be <http://www.w3.org/XSL/Transform/1.0>.

```

<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"
  xmlns="http://www.w3.org/TR/REC-html40" result-ns="">

  <xsl:template match="/">
  <HTML>
    <HEAD>
      <TITLE>XML Formatting Servlet</TITLE>
    </HEAD>
    <BODY>
      <H1>XSL Processing Example</H1>
      <xsl:apply-templates/>
    </BODY>
  </HTML>
</xsl:template>

<xsl:template match="employees">
  <TABLE border="1" frame="border" rules="all" cellpadding="2">
    <CAPTION class="listingCaptions">Phone List</CAPTION>
    <COLGROUP>
      <COL width="30" align="right"/>
      <COL padding-left="15"/>
      <COL padding-left="15"/>
      <COL padding-left="15" align="center"/>
    </COLGROUP>
    <THEAD class="tableHead">
      <TD>Name</TD>
      <TD>Phone</TD>
      <TD>Title</TD>
      <TD>Department</TD>
    </THEAD>
    <xsl:for-each select="employee">
      <!-- The '.' context is now the "employee" element -->
      <xsl:sort select="name/@last" order="ascending" data-type="text"/>
      <xsl:sort select="name/@midinit" order="ascending" data-type="text"/>
      <xsl:sort select="name/@first" order="ascending" data-type="text"/>
      <TR>
        <TD class="cell">
          <!-- process the text from the "name" element -->
          <xsl:apply-templates select="name"/>
        </TD>
        <TD class="cell">
          <!-- get the text from the "phone" element -->
          <xsl:value-of select="phone"/>
        </TD>
        <TD class="cell">
          <!-- get the text from the "title" element -->
          <xsl:value-of select="title"/>
        </TD>
        <TD class="cell">
          <!-- process the text from the "birthdate" element -->
          <xsl:value-of select="@department"/>
        </TD>
      </TR>
    </xsl:for-each>
  </TABLE>
</xsl:template>
<xsl:template match="name">
  <xsl:value-of select="@last"/>
  <xsl:text>, </xsl:text>
  <xsl:value-of select="@first"/>
  <xsl:text> </xsl:text>
  <xsl:value-of select="@midinit"/>
</xsl:template>
</xsl:stylesheet>

```

Figure 150. Phonelist.xsl

- On the next line we define the default namespace, the one that applies if we specify no prefix on the tags, to be the HTML 4.0 namespace. This line also declares the result namespace to take the default value that is the same as the default namespace. See "XML Namespaces by Example" by Tim Bray at <http://www.xml.com/xml/pub/1999/01/namespaces.html> for more information on understanding XML namespaces.

- Next we declare an `xsl:template` that matches `"/`. Just like in a file system, the `"/` matches the root element of any XML tree. In our case this is a convenient place to specify some boilerplate HTML that will serve as a wrapper for all of the other HTML that we will produce. Note also that within this template we have an XSL directive:

```
<xsl:apply-templates/>
```

which is an instruction for the XSL processor to look at all of the descendants of the current element, in this case the root element of the XML tree, and match any templates that apply. If this instruction were left out, the XSL processor would stop processing after producing the text for the root element and not apply the other tags.

- The next step is to declare a template that matches the `"employees"` element, which as you will remember is the top level element in our XML document. Within the `employees` element we produce the outer definitions for the HTML table that will hold our data. Within this table definition there is another XSL processing instruction:

```
<xsl:for-each select="employee">
```

which requires the end tag:

```
</xsl:for-each>
```

further down in the code. This tells the XSL processor to look for each node in the XML source tree that matches the pattern contained in the `select` attribute and for each of them replace them with the text contained within the `xsl:for-each` tag. See the June 1999 issue of *Web Techniques Magazine* at <http://www.webtechniques.com> for Michael Floyd's column "Beyond HTML" for a good tutorial on how to write XSL patterns. In this case the pattern will match employee element children of the current node that is the `employees` element.

- Within the `xsl:for-each` tag we have the code to generate an individual table row. The first thing that we see is a group of `xsl:sort` directives:

```
<xsl:sort select="name/@last" order="ascending" data-type="text"/>
<xsl:sort select="name/@midinit" order="ascending" data-type="text"/>
<xsl:sort select="name/@first" order="ascending" data-type="text"/>
```

These have the effect of telling the XSL processor to sort the output of the `xsl:for-each` tag by certain criteria before producing the output. The criteria are specified by specifying a pattern that is evaluated for each node processed by the `xsl:for-each` directive and then sorted according to the other criteria. In this case we specify that the sorting is to be first by the `"last"` attribute of the `"name"` element that is a child of the current `employee` element, and then within that, by the `"midinit"` and `"first"` attributes. In each case the sorting is to be ascending and based on the text collation order of the current language and character set. The effect of these directives is to sort by last name, then within that to sort by middle initial, and then within that to sort by first name.

The next part of the `xsl:for-each` tag is the HTML commands to generate the cells in the table row for this `employee` element. You will notice a number of XSL directives similar to the following:

```
<xsl:value-of select="phone"/>
```

These have the effect of taking any text that is included between the start and end tags of the element specified and inserting it in the results. We could have also specified a pattern resolving to an attribute here, in which case the attribute value would have been inserted. This allows us to insert data from

the XML into our output rather than just using the node structure to generate output as we have been doing up until now.

The other XSL processing directive that is included in the cell definitions is the directive to process the name tags:

```
<xsl:apply-templates select="name"/>
```

This is similar to the processing directive that we saw in the template for the root element above, except this time we specify a pattern for the nodes that the XML processor will look at in order to apply the templates. In our case we only want to process the name element children of the current employee element at this time so we use `select="name"`.

- The last element in the XSL file is another template, this time one to match the name element. In the template above we included an `xsl:apply-templates` directive to match the name elements and this is where the information for each name element will be produced. `xsl:value-of` directives are used to get the attribute values of the name element (note the "@" signs denoting attributes). The other new directive is the `xsl:text` directive, which is used to tell the XSL processor to insert some text into the output stream. Up until now we have been producing HTML tagged data that conforms to the default namespace we declared at the beginning of the file:

```
xmlns="http://www.w3.org/TR/REC-html40"
```

If we want the XSL processor to produce text in the output that is not part of a tag we need to use the `xsl:text` directive. The effect of these directives is to cause the last name to be output followed by a comma and a space followed by the first name followed by a space followed by the middle initial.

The birthday list style sheet, `birthday.xsl` (see Figure 151) is similar to the phone list style sheet, but it implements a different view of the data, showing a list of birthdays:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"
  xmlns="http://www.w3.org/TR/REC-html40" result-ns="">

  <xsl:template match="/">
    <HTML>
      <HEAD>
        <TITLE>XML Formatting Servlet</TITLE>
      </HEAD>
      <BODY>
        <H1>XSL Processing Example</H1>
        <xsl:apply-templates/>
      </BODY>
    </HTML>
  </xsl:template>

  <xsl:template match="employees">
    <TABLE border="1" frame="border" rules="all" cellpadding="2">
      <CAPTION class="listingCaptions">Birthday List</CAPTION>
      <COLGROUP>
```

Figure 151. `Birthday.xsl`

The following differences are worth noting:

- In this style sheet we are sorting by the birth month and then the birth day rather than by the name of the person.
- The template that we are using for the name is slightly different, showing the first and last name separated by a space.

- We have a new template for the birthdate producing the birth day followed by a "/" followed by the birth month.

The seniority list style sheet, `seniority.xml` (see Figure 152) is similar to the other style sheets. It implements a view of the data, showing a list of employees with the last hired shown first and the longest-serving employees shown last.

```

<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"
  xmlns="http://www.w3.org/TR/REC-html40" result-ns="">

  <xsl:template match="/">
    <HTML>
      <HEAD>
        <TITLE>XML Formatting Servlet</TITLE>
      </HEAD>
      <BODY>
        <H1>XSL Processing Example</H1>
        <xsl:apply-templates/>
      </BODY>
    </HTML>
  </template>

```

Figure 152. `Seniority.xml`

The following differences are worth noting:

- The sort order in this style sheet is again different. Note that descending sort order is used as well as numeric rather than text-based sorting.
- The hiredate template rearranges the order of the data fields so that the year comes first, followed by the month followed by the day.

3.3.6.5 Building a Prototype and Fixing Problems

The next step in the development process was to hunt around for example code from the various tools used and build a non-servlet prototype using VisualAge for Java. This was useful, because it allowed the testing of the logic in an easily debuggable environment away from any considerations introduced by WebSphere. It also uncovered a number of errors in the logic as the program was developed. The first stage was to use the hand generated XML to produce an HTML file on the development machine and then to gradually introduce new features till the code was extracting data from the database and producing HTML to standard output. This process uncovered a number of things:

- XML is a complex and evolving field that necessitated much trial and error. Hopefully this code will help to iron out some of the pitfalls, but the moment that you try something a little different you will need to look for resources to help you fix the problems. The following is a list of the resources that we used during development and that you might find useful:
 - The annotated XML spec at <http://www.xml.com/axml/testaxml.htm>. There's no substitute for the specification and this version makes things a little easier to understand.
 - The rest of the <http://www.xml.com> site.
 - The IBM XML site, which at the time of writing was being moved to <http://www.ibm.com/developer/xml/>.

- The WebSphere newsgroup at news:news.software.ibm.com/ibm.software.websphere.application-server.
- The alphaWorks LotusXSL and XML4J discussion groups at <http://www.alphaWorks.ibm.com>.
- It always helps to run your stand-alone code outside of your development environment before trying to run it as a servlet in WebSphere. The release of LotusXSL or XML4J that we used didn't like the JIT compiler on either of the JDKs that we used for testing (IBM JDK 1.1.7 and Sun JDK 1.1.6). See 3.3.6.10, "Description of Errors Encountered during Development" on page 181 for a full description of the symptoms encountered. It was only after running the code as a stand-alone program from the command line that we were able to find this problem.

3.3.6.6 The Servlet Code

Now that we have all of the data pieces together, it is time to have a look at the servlet code that ties everything together. Figure 153 shows the source code for XSLServlet.java:

```
package com.ibm.pjk.xml;

import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.io.*;
import java.sql.*;
import com.ibm.xml.parsers.*;
import org.xml.sax.*;
import com.lotus.xsl.*;
import com.lotus.xsl.xml4j2dom.*;
/**
 * This type was created in VisualAge.
```

Figure 153. XSLServlet.java

The following points about the code are worth noting:

- There are a number of output statements in the code that produce informational output for debugging purposes. They have been left in to show what the code is doing. If this code is modified to be used in a production environment, then remove these statements. In particular the DB2 JDBC driver log output is redirected to standard output and the XML generated from the DB2 query is redirected to standard output.
- The DB2 driver is registered in the static block at the start of the code to ensure that it is registered once when the class is first loaded.
- The constructor does nothing other than to call the superclass constructor.
- Among the variable declarations there are a number of machine specific parameters:

```
//Machine specific parameters - change these for your environment
String userid = "peterk";
String password = "xxxxxxx";
String machineName = "wtr05073.itso.ral.ibm.com";
String dtdPath = "/employee.dtd";
```

These parameters need to be changed for your environment if you want to run this code. Note that the user ID and password were ones used during development and bear no resemblance to real user IDs and passwords used on production systems. The machineName value is the name of your Web server. Change the dtdPath value to reflect the location of your DTD document on your Web server.

- The code uses a style sheet parameter value passed to it by the HTTP request to determine which style sheet to use. This parameter contains the complete URL of the style sheet.
- The technique used to generate the XML and then to process it using LotusXSL was to first write it to a byte array using a `ByteArrayOutputStream`. This byte array could then be used as input to an `InputStreamReader` via a `ByteArrayInputStream` to provide the reader argument for the LotusXSL processor.
- The database is queried using JDBC calls. For more information on using JDBC see 6.1, “JDBC” on page 332.
- Before adding the data for each employee the XML header information, including the namespace declarations and the style sheet definition, are written to the output stream. This is followed by the start tag for the document type: `<employees>`.
- For each row of the database the information contained in the result set for the query is extracted into a number of string variables for later processing.
- The hiredate and birthdate values returned from the database are in the form of `java.sql.Date` values. To extract the year, month, and day values, each `Date` value is inserted into a `GregorianCalendar` object using the `setTime` method and then the individual values are extracted into an array using the `GregorianCalendar` `get` method with the appropriate constant specifying which value we want to get. The array values are then used later to construct the XML.
- Next the XML is written to the output stream using the data values extracted from the result set. This format follows closely that of the hand-created XML shown in Figure 149 on page 169. Tabs are inserted in the output so that it is formatted nicely for debugging purposes. Their omission would not change the semantic content of the XML.
- After all of the employee data has been written, the end tag for the employees element is written and the output stream is closed and flushed.
- The next step is to prepare the response header fields and to extract the print writer from the response object so that it can be used to write the HTML. The response object is set up to disable caching on the client so that the dynamically generated data is regenerated each time the servlet is called rather than being taken from a cache.
- After setting up a writer to read the XML from the byte array created earlier it is time to do the XSL processing. The first step is to create an instance of the XSL processor. This needs to be done each time a style sheet is processed; you need a new instance for each operation. The argument to the constructor is an instance of the `XMLParser LiasonDefault` class, which is used by LotusXSL to provide an interface between the XSL processor and the XML4J parser.

- The final step is to call the process method with the reader for the XML as the first argument and the writer from the response object, to which the HTML will be written, as the second argument.
- The rest of the code is some error handling to print informative messages if something should go wrong and a final clause that flushes and closes the response object to make sure that all of the output is written.

3.3.6.7 Creating HTML to Run the Servlet

The final step in the process was to create some framework HTML to run the servlet. Since this was only to test the servlet code, the HTML we wrote was very basic. Figure 154 shows the HTML we used to test the servlet:

```
<HTML>
<HEAD>
<TITLE>WebSphere Samples</TITLE>
</HEAD>
<BODY>
  <form Method="get" ACTION="/servlet/XSLServlet">
    <font size=-1>StyleSheet URL</font>&nbsp;
    <input type="text" name="stylesheet" size=30>
    <br><br>
    <input type="submit" value="Submit">
  </form>
</BODY>
```

Figure 154. XSLTest.html

The HTML provides a text entry field in which to provide the URL for the style sheet to use and a button to submit the data and call the servlet.

3.3.6.8 Putting It All Together

Once all of the pieces have been created it is time to put them all together. The following tasks need to be performed:

1. Copy the DTD, XSL and HTML files on the Web server.

The following files need to be placed in a directory on the Web server accessible via HTTP:

```
employee.dtd
phonelist.xsl
birthday.xsl
seniority.xsl
XSLTest.html
```

In our case, using the IBM HTTP server installed on Windows NT on drive D meant putting them in the d:\Program Files\IBM HTTP Server\htdocs directory.

2. Compile the Java program and put it in the WebSphere Servlets directory.

The Java class file for the servlet needs to be compiled and placed in the correct directory so that the WebSphere servlet engine can find it. In our case we used the WebSphere default servlet directory which meant that the class file had to be put in the <ASRoot>\servlets\com\ibm\pjk\xml directory according to the package name we used: com.ibm.pjk.xml.

3. Register the servlet to WebSphere.

The next step is to register the servlet to WebSphere so that it can be called by a client. The procedure for adding servlets can be found in 3.1.3.2, “Adding Servlets” on page 97. We added the servlet with an alias of XSLServlet.

4. Ensure that DB2 is running and the sample database has been created.

Because this sample uses the DB2 sample database, you need to ensure that the DB2 database instance you will access is running and that the sample database has been created. See your DB2 documentation for instructions on how to create the sample database. For Windows NT you can type `db2samp1` in a DB2 command window to create the sample database.

5. Run the servlet using the test HTML page.

Figure 155 on page 178, Figure 156 on page 179, and Figure 157 on page 180 show the HTML pages produced as a result of running the XSL servlet with the phone list, birthday and seniority list style sheets respectively. To do this we input the URLs of the style sheets on our Web server into the input field on the test HTML page and clicked **submit**.

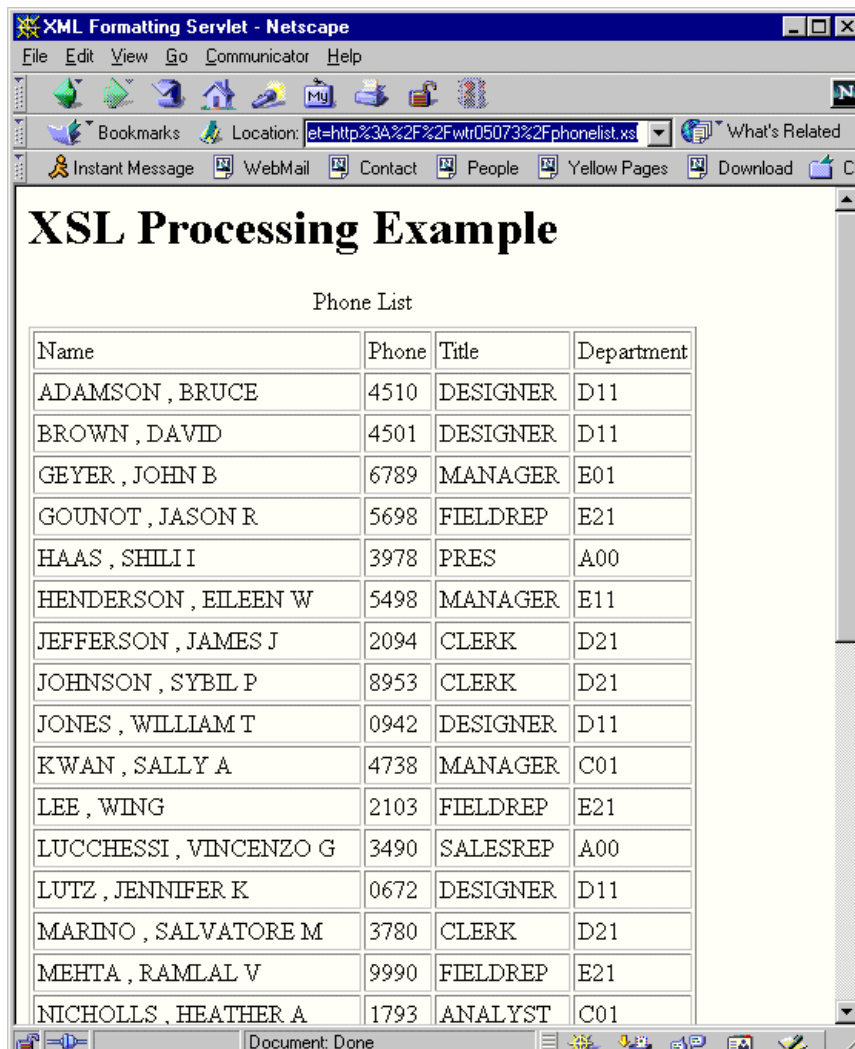
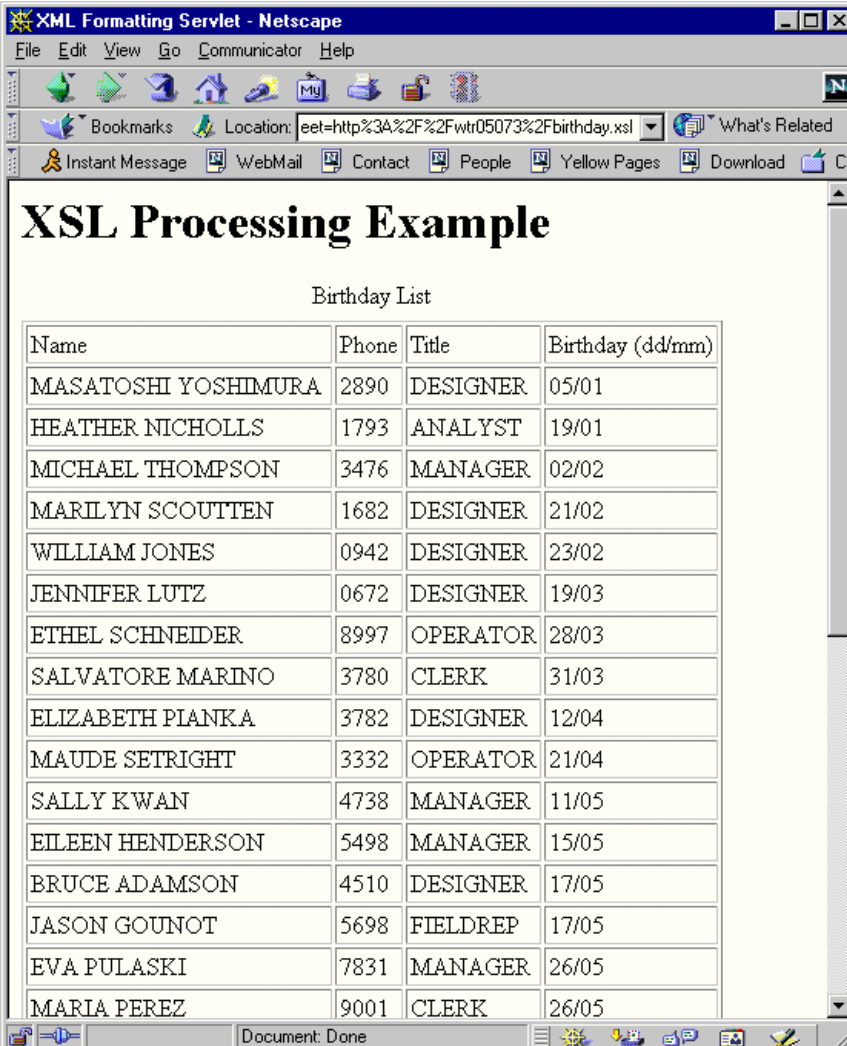


Figure 155. Results of Running the XSL Servlet with `phonelist.xml`

Note that the XML that we generate from the database is identical in every case except for the line that specifies the style sheet. The style sheet instructions as interpreted by the XSL processor do the rest.



XSL Processing Example

Birthday List

Name	Phone	Title	Birthday (dd/mm)
MASATOSHI YOSHIMURA	2890	DESIGNER	05/01
HEATHER NICHOLLS	1793	ANALYST	19/01
MICHAEL THOMPSON	3476	MANAGER	02/02
MARILYN SCOUTTEN	1682	DESIGNER	21/02
WILLIAM JONES	0942	DESIGNER	23/02
JENNIFER LUTZ	0672	DESIGNER	19/03
ETHEL SCHNEIDER	8997	OPERATOR	28/03
SALVATORE MARINO	3780	CLERK	31/03
ELIZABETH PIANKA	3782	DESIGNER	12/04
MAUDE SETRIGHT	3332	OPERATOR	21/04
SALLY KWAN	4738	MANAGER	11/05
EILEEN HENDERSON	5498	MANAGER	15/05
BRUCE ADAMSON	4510	DESIGNER	17/05
JASON GOUNOT	5698	FIELDREP	17/05
EVA PULASKI	7831	MANAGER	26/05
MARIA PEREZ	9001	CLERK	26/05

Figure 156. Results of Running the XSL Servlet with birthday.xsl

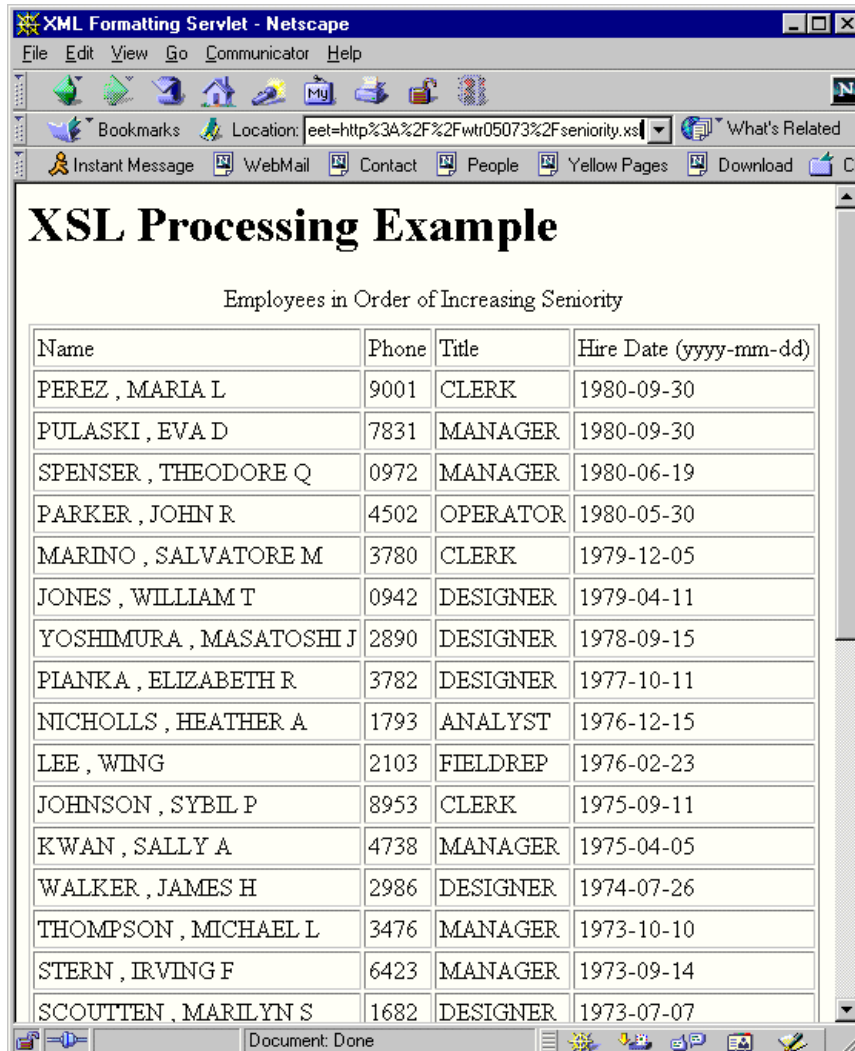


Figure 157. Results of Running the XSL Servlet with `seniority.xml`

3.3.6.9 Possible Improvements and Changes

The following is a list of possible improvements or changes you might like to make to the code when implementing it in your environment:

- The parameters that specify such things as the database user ID and password are hard coded into the `XSLServlet.java` file. For more general use these should be read from a `.properties` file somewhere.
- Now each time the servlet is called it must read information from the database and regenerate the XML. It would be better if this XML could be cached somehow and only regenerated if the database changed. The only change to be made would be the style sheet reference.
- The HTML files generated by the XSL processor are extremely basic. They could easily be modified to look much better through the use of images, color and cascading style sheets (cascading HTML style sheets have nothing to do with XSL style sheets).
- At the moment the HTML produced is for a complete HTML page. In a production environment it may be desirable to have the servlet only produce

the table portion of the HTML and have another part of the application, a JSP page for instance, produce the rest of the page.

- The code could be easily adapted to read from other database tables or even other database engines that support JDBC.
- The style sheets used only begin to scratch the surface of what can be accomplished with XSL transformations. They could be modified to perform more complex transformations on the XML to achieve more sophisticated results.

3.3.6.10 Description of Errors Encountered during Development

During the development process we encountered a number of errors and problems in completing the task. They ranged from the trivial to discovering a potential bug in the tools themselves. There were two errors in particular that proved difficult to track down from the information provided by various diagnostic messages and logs. These two errors are described here in the hope that the descriptions will save some time if you ever come across them.

1. Can't find the class for servlet XSLServlet: null

When you are adding a servlet there is an option for WebSphere to try and test the loading of the servlet class prior to completing the add. If, during this test, you get a message box similar to the one shown in Figure 158, it usually means that WebSphere cannot find a class associated with your servlet and that you should check your classpath settings.

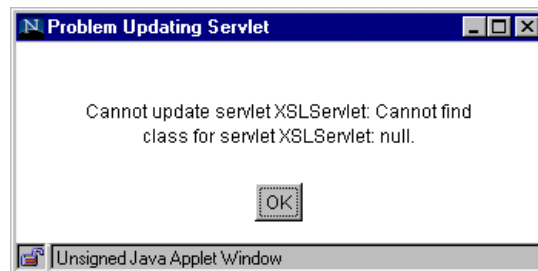


Figure 158. Servlet Test Error Box

In this case, however, the place where WebSphere usually puts the name of the class it cannot find has the value null, which is most unhelpful. This null value is WebSphere's way of telling us that it cannot find the class itself, even though it can find the file that the class is in. If it could not find the class file it would give us another message.

The solution to this problem is to go back to your source code and make sure that your servlet class is declared as `public` so that WebSphere can find it.

2. Running the XSLServlet causes the WebSphere Servlet Service to hang.

The symptom of this problem was that a servlet using the XML parser would produce the error dialog shown in Figure 159 under Netscape Navigator (other browsers would have similar messages).



Figure 159. Netscape Error Dialog Box

Attempting to reload the servlet at this point would produce the error page from WebSphere shown in Figure 160:



Figure 160. WebSphere Servlet Service Unavailable Page

Attempting to stop and restart WebSphere would produce the following message in the <ASRoot>\logs\servlet\adminservice\error_log:

```
[May 17, 1999 3:45:05 PM EDT] Cannot start service adminservice: Port 9527 is already in use. Change the port or shut down the other network service that is using it.
```

The Windows NT Task Manager would show four java.exe processes that would give the error dialog box shown in Figure 161 when the **End Process** button was used to attempt to terminate them:

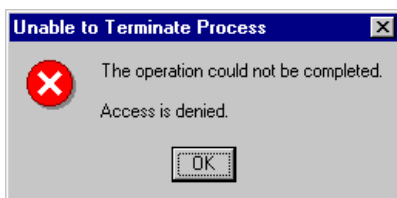


Figure 161. Windows NT Process Termination Error

Paths	Memory	Directives	Process						
Java Compiler: <input type="text" value="-nojit"/>									
Library Settings <table border="1"> <tr> <td>Java Normal Library:</td> <td><input type="text" value="javai.dll"/></td> </tr> <tr> <td>Java Debug Library:</td> <td><input type="text" value="javai_g.dll"/></td> </tr> </table>				Java Normal Library:	<input type="text" value="javai.dll"/>	Java Debug Library:	<input type="text" value="javai_g.dll"/>		
Java Normal Library:	<input type="text" value="javai.dll"/>								
Java Debug Library:	<input type="text" value="javai_g.dll"/>								
Garbage Collection <table border="1"> <tr> <td>Perform Class Garbage Collection?</td> <td><input checked="" type="radio"/> Yes</td> <td><input type="radio"/> No</td> </tr> <tr> <td>Asynchronous Garbage Collection?</td> <td><input checked="" type="radio"/> Yes</td> <td><input type="radio"/> No</td> </tr> </table>				Perform Class Garbage Collection?	<input checked="" type="radio"/> Yes	<input type="radio"/> No	Asynchronous Garbage Collection?	<input checked="" type="radio"/> Yes	<input type="radio"/> No
Perform Class Garbage Collection?	<input checked="" type="radio"/> Yes	<input type="radio"/> No							
Asynchronous Garbage Collection?	<input checked="" type="radio"/> Yes	<input type="radio"/> No							

Figure 163. The WebSphere Java Engine Directives Page

3.3.6.11 Release 0.17.0 of LotusXSL

We also tried running the code under Release 0.17.0 of LotusXSL with limited success. This release of LotusXSL conforms to the April 21, 1999 version of the XSLT specification which can be found at

<http://www.w3.org/TR/1999/WD-xslt-19990421.html>. The only change that we made to the code was to add a different namespace URL to the XSL style sheets, replacing <http://www.w3.org/TR/WD-xsl> with <http://www.w3.org/XSL/Transform/1.0>.

The results of running the birthday.xsl style sheet (see Figure 151 on page 173) with the later release are shown in Figure 164.

We were unable to determine if the duplicated table header was the result of a change in the XSL specification that necessitated a change in our XSL style sheet or a bug in LotusXSL.

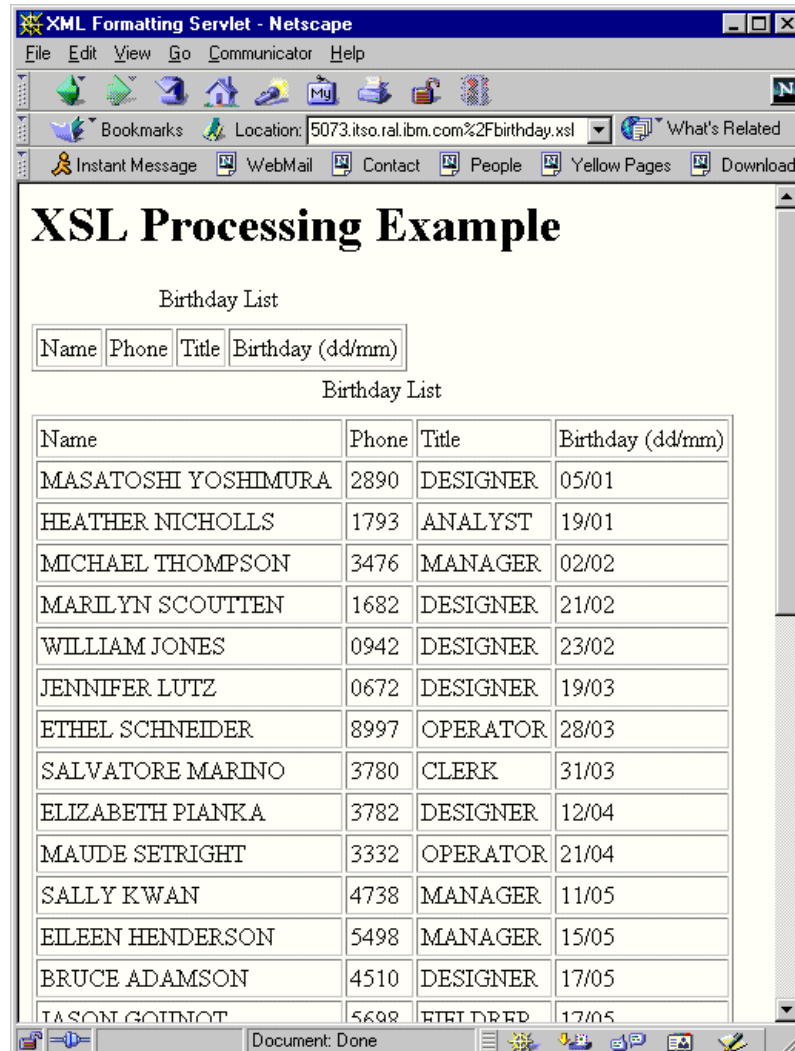


Figure 164. Results of Running birthday.xsl with LotusXSL V0.17.0

3.3.7 Installing Later Versions of the XML Tools

So far most of this section has been confined to discussing the version of the XML document Structure Services shipped with WebSphere 2.02. These services are based on a version of the IBM XML for Java parser and on a version of the LotusXSL style sheet processor. Newer versions of these technologies are available from the IBM alphaWorks Web site at <http://www.alphaworks.ibm.com> and these can be installed in place of the versions shipped with the product.

To install a newer version, download the new distribution file and unpack it according to the download instructions, then perform the following steps:

1. Locate the JAR files containing the product APIs.

These files will probably be located in the top level directory to which you unpacked the tools. For the IBM XML for Java parser this file will be called xml4j.jar. For LotusXSL it will be called lotusxsl.jar.

2. Make backup copies of the original product files.

The JAR files shipped with the original product are located in the directory <ASRoot>\lib. Copy the files xml4j.jar and lotusxsl.jar to a safe location to enable you to back out the changes if there are problems. You may need to stop the WebSphere servlet engine in order to release the locks on the files so that you can accomplish this. On Windows NT go to **Start->Settings->Control Panel**, then double-click **Services**. When the services window opens, scroll down and select the **WebSphere Servlet Service** and click **Stop**.

3. Copy the new files to the WebSphere lib directory.

Copy the xml4j.jar and lotusxsl.jar files that you located in step 1 to the <ASRoot>\lib directory in place of the original files.

4. Stop and restart the Web server to pick up the changes.

An alternative to the above procedure would be to add the xml4j.jar and lotusxsl.jar files to the application server classpath in place of the ones shipped with the original product. The procedure to add files to the classpath is covered in 3.3.2, "Setting Up the Environment" on page 154.

WebSphere 2.02 ships with Version 1 of the IBM XML parser for Java. The latest version available when this book was being written was 2.0.6. There have been significant changes to the API between these two releases, so a careful study of the documentation is required in order to understand if any change in applications developed with the older APIs is desirable or necessary. In particular there has been a new package added: *com.ibm.xml.parsers* (as opposed to *com.ibm.xml.parser*), which includes a number of parser variants that conform more closely to the W3C standard and have better performance. These parsers do not have all of the functionality of the older APIs. The Version 1 APIs are still present in the product and they are referred to as the *TX Compatibility* APIs in the documentation.

If installing a newer version of LotusXSL it should be noted that there are dependencies between LotusXSL and the IBM XML for Java parser. These dependencies are detailed in the LotusXSL documentation.

The WebSphere XML samples were tested with the IBM XML for Java parser Version 2.0.6 and the event-driven parsing samples. The SAX parser sample, and the sample on parsing using event handlers, did not work for us. Please check the documentation for any new version that is available to install.

Chapter 4. Enterprise Java Services

Note: Enterprise Java Services are not included with WebSphere Application Server 2.X Standard Edition. These features are included only in the Advanced Edition.

Enterprise Java Services, or EJS, is the WebSphere component that allows the deployment and management of Enterprise Java Beans in EJB containers. It is in fact made up of a number of interrelated pieces including a naming service, EJB containers and management services. In this chapter we will look at how to work with EJS, including how to install and run the WebSphere EJS samples.

What we will not look at in this chapter is how to develop EJBs or EJB clients. This is beyond the scope of this book. Where appropriate, references have been made in this chapter to reference materials relating to EJB development. Readers are encouraged to review the EJB overview presented in 1.3.5, “Enterprise Java Beans” on page 6.

A more in-depth technical overview of EJB development can be found at <http://java.sun.com/products/ejb/developers-guide.pdf>.

4.1 The EJS Java Processes

EJS runs three processes on your WebSphere system when enabled. The Persistent Naming Service (PNS) and Location Service Daemon (LSD) processes are used by EJS to provide naming services to EJB clients so that they can locate EJBs deployed on the server. The EJS Server process is where the EJB containers and management services run. Each of these processes runs in a separate Java Virtual Machine on the server. This is in addition to the servlet service process that also runs in its own virtual machine.

Figure 165 shows the results of running `ps -ef | grep java` on an AIX system to display the Java processes running on the machine. The first process is the servlet process followed by the LSD, PNS and EJS server processes and finally the `grep` command itself.

```
# ps -ef | grep java
nobody 24510 1 1 May 14 pts/2 118:17 java
com/ibm/servlet/engine/outofproc/OutOfProcEngine -nativelogfile
/usr/WebSphere/AppServer/logs/oop_native.log -nativeloglevel 14 -linktype local -port 8081
-queueaname ibmappserve -stublib /usr/WebSphere/AppServer/plugins/aix/libosestub.so -serverlib
/usr/WebSphere/AppServer/plugins/aix/libasouts.so
nobody 26060 24510 0 May 14 pts/2 0:43 java com/ibm/servlet/ejs/LSDInstance 9091
controller 9030 LSD localhost 9029
nobody 26332 24510 0 May 14 pts/2 0:44 java com/ibm/servlet/ejs/PNSInstance 9091
controller 9028 PNS localhost localhost 9029 /usr/WebSphere/AppServer//temp 9019
nobody 26596 24510 0 May 14 pts/2 51:52 java com/ibm/servlet/ejs/EJSInstance 9091
controller 9027 EJS localhost 9029 /usr/WebSphere/AppServer//properties/ejs/ejs.properties
9019 /usr/WebSphere/AppServer/ null
root 29370 21424 1 16:08:01 pts/1 0:00 grep java
```

Figure 165. Java Processes on an AIX System Showing the Three EJS Processes

On Windows NT it is more difficult to tell the four Java processes apart. Figure 166 on page 188 shows a screen from Windows NT task manager. The four java.exe processes shown in the middle of the screen belong to WebSphere.

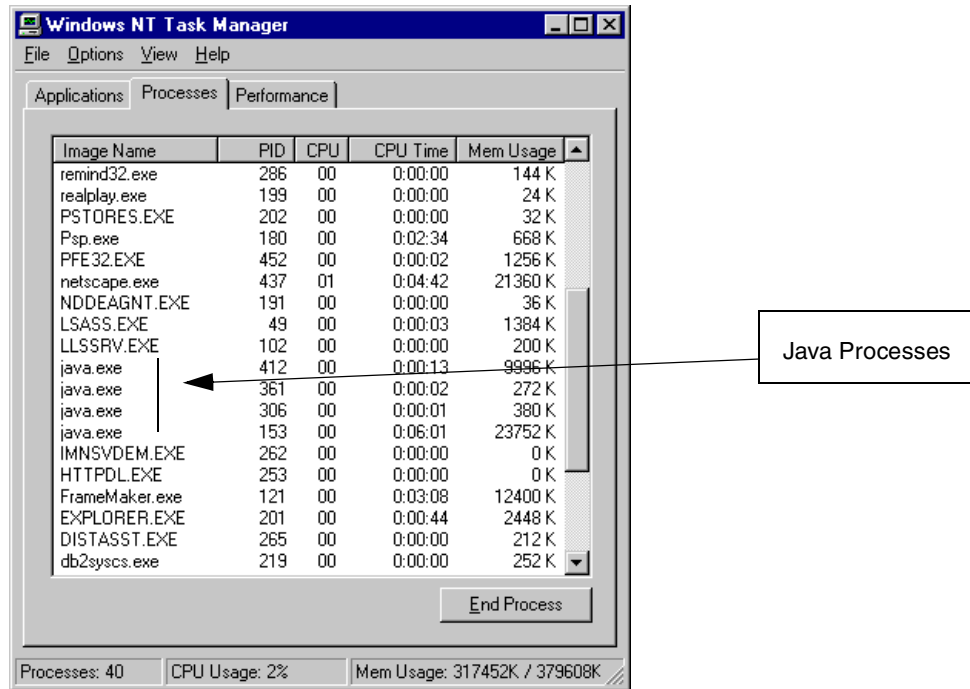


Figure 166. Windows NT Task Manager Showing the Four WebSphere Java Processes

4.2 Configuring Enterprise Java Services

This section shows the steps required to configure EJS and deploy EJBs in WebSphere Application Server. Most of these procedures assume that you are logged on to the WebSphere Application Server Administration Interface using your administrator user ID and password. This can be achieved by opening the Web address `http://<your server name>:9527/` and typing your user ID and password in the fields provided.

4.2.1 Setting Up the Environment

Most of the environment setup for EJS is covered in the sections on working with containers and deploying EJBs, but there are a few other parameters that affect how the EJS services will run.

Figure 167 shows the entry field area of the EJS global settings page that can be reached by logging on to WebSphere Administration and then selecting **Enterprise Java Services -> Global Settings**.

Figure 167. The EJS Global Settings Page

Table 15 shows the fields and their possible values:

Table 15. Field Values for EJS Configuration

Field Name	Description
Enable EJS	This field allows EJS to be enabled or disabled. EJS is enabled by default.
EJS Server Name	This is the JNDI name of the WebSphere EJS server. By default this is set to server1.
Remote Access	This parameter relates to whether or not remote clients have access to the EJS server.
Enable EJS Tracing	This parameter governs whether or not the tracing output produced by the EJS engine is produced. This overrides any setting for EJS_stderr and EJS_stdout on the Server Execution Analysis -> Trace page. By default this value is set to on.
Host Name	The name of the host providing the naming service.
Boot Port	The port number for the EJS naming service. By default this value is set to 9019.
Protocol	The protocol used by the naming service. There is no indication as to whether anything other than CORBA can be placed in this field or what the effect would be.

4.2.2 Working with Containers

In order to deploy EJBs on a server a container is required. Containers are the basic service provided by an EJB server. Each EJB exists in a container that provides certain services and facilities to the EJB that are defined in the EJB specification. This is known as the container contract. Each EJB container provider is responsible for implementing container classes that provide the services specified in the container contract to EJBs deployed in the containers.

Containers also provide the mechanisms by which clients interact with EJBs, controlling and filtering access to the EJBs contained within.

WebSphere allows the creation of multiple containers on the server, each with a separate database connection, database user ID, name and separate deployment directories. Each container may also be configured for use by either session or entity EJBs through the specification of the Java class file used to implement the container.

Each EJB JAR file may be deployed in single or multiple containers depending on what particular connections it requires.

It is a good idea to delete containers that have no EJBs deployed in them as this can cause errors in the EJS naming service.

4.2.2.1 Creating a Container

Perform the following steps to create a new EJB container:

1. In the left-hand column of the administration interface expand the choices under **Enterprise Java Services** by clicking the *twistie* (the small triangle) and select **Containers** (see Figure 168 on page 190).

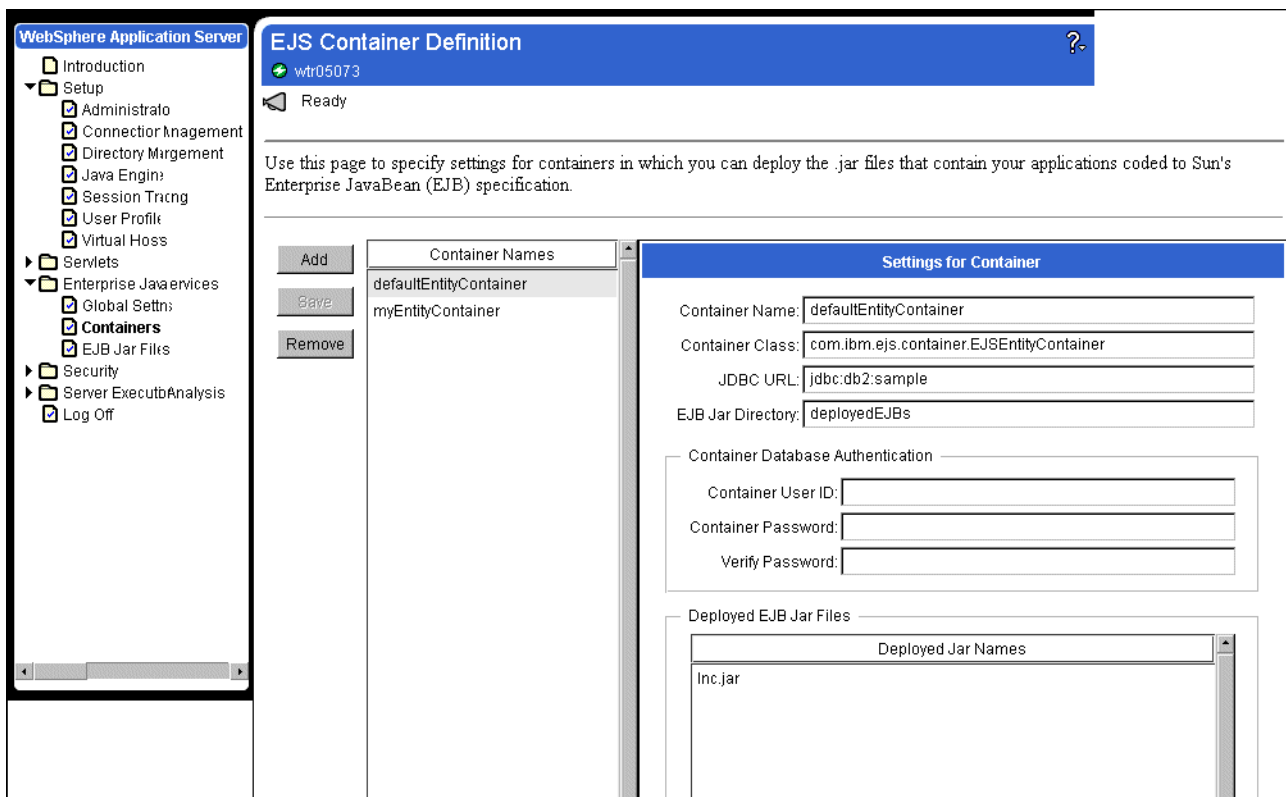


Figure 168. The EJS Container Page

2. Click **Add** and fill in the name of the new container in the Add a New Container window (see Figure 169 on page 191):

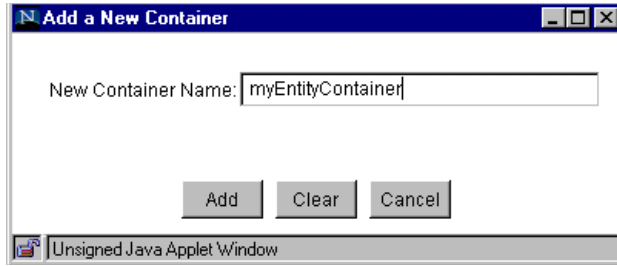


Figure 169. Adding a New EJB Container

3. Enter values in the fields on the container page for your new container (see Figure 168 on page 190) according to the values in Table 16. If using Netscape Navigator, do not resize the browser window before completing the next step (to save your data), as this will cause any changes to be lost.

Table 16. Parameter Values for Container Fields

Parameter	Possible Value	Comments
Container Name	Anything you choose	Name used to identify the container to WebSphere.
Container Class	com.ibm.ejs.container.EJSEntityContainer or com.ibm.ejs.container.EJSSessionContainer	Name of the class implementing the container function. Use EJSEntityContainer for containers to hold entity EJBs and EJSSessionContainer for containers to hold Session EJBs.
JDBC URL	Any JDBC URL. For DB2 this might be something like jdbc:db2:ejs_samp	JDBC driver URL for the database used to store EJB information. This value is only required for containers with Entity EJBs with Container Managed Persistence. If the EJBs to be deployed need different databases, create multiple containers, each with the URL of the desired database.
EJB JAR Directory	Relative directory name	Directory relative to <server root> where EJBs are to be deployed. This value defaults to DeployedEJBs.
ContainerUser ID	Your database logon ID	Database logon ID for the container to use when it accesses the database. This value is only required for containers with Entity EJBs with Container Managed Persistence.
Container Password	Your database password	Database password for the container to use when it accesses the database. This value is only required for containers with Entity EJBs with Container Managed Persistence.

4. Click **Save** to save your changes.
5. Stop and restart the EJS processes to activate the changes.

On Windows NT click **Start -> Settings -> Control Panel -> Services** and find the WebSphere Servlet Service. Stop this service by selecting it and clicking **Stop**. That will stop the EJS processes. Next stop and restart your Web server to restart EJS.

On AIX you will need to stop the Web server and then kill all of the WebSphere processes before restarting the Web server (see Figure 165 on page 187).

4.2.2.2 Removing a Container

In order to remove a container, perform the following steps:

1. In the left-hand column of the administration interface expand the choices under **Enterprise Java Services** by clicking the twistie (the small triangle) and select **Containers** (See Figure 168 on page 190).
2. Under **Container Names** select the name of the container you wish to remove and click **Remove**.
3. A dialog box asks you to confirm the delete. Click **Yes** to delete the container.
4. Stop and restart the EJS processes to activate the changes.

On Windows NT click **Start -> Settings -> Control Panel -> Services** and find the WebSphere Servlet Service. Stop this service by selecting it and clicking **Stop**. This will stop the EJS processes. Next stop and restart your Web server to restart EJS.

On AIX you will need to stop the Web server and then kill all of the WebSphere processes before restarting the Web server (see Figure 165 on page 187).

4.2.2.3 Setting Container Parameters

If you need to change the parameters for an existing container, perform the following steps:

1. In the left-hand column of the administration interface expand the choices under **Enterprise Java Services** by clicking the twistie (the small triangle) and select **Containers** (See Figure 168 on page 190).
2. Under Container Names select the name of the container you wish to modify.
3. Edit the container fields according to Table 16 on page 191 and click **Save**. If you are using Netscape Navigator, do not resize the browser window until after saving your changes or they will be lost.
4. Stop and restart the EJS processes to activate the changes.

On Windows NT click **Start -> Settings -> Control Panel -> Services** and find the WebSphere Servlet Service. Stop this service by selecting it and clicking **Stop**. This will stop the EJS processes. Next stop and restart your Web server to restart EJS.

On AIX you will need to stop the Web server and then kill all of the WebSphere processes before restarting the Web server (see Figure 165 on page 187).

4.2.3 Deploying an EJB

4.2.3.1 Deploying an EJB in a Container

In order to deploy an EJB in a container perform the following steps:

1. Copy the EJB JAR file containing the EJB or EJBs into the <Server Root>\deployable EJBs directory. See 4.2.3.3, “WebSphere Generated Classes” on page 196 for an explanation of the files required in an EJB JAR file.
2. In the WebSphere administration interface select **Enterprise Java Services** and then **EJB JAR Files** (see Figure 170 on page 193).

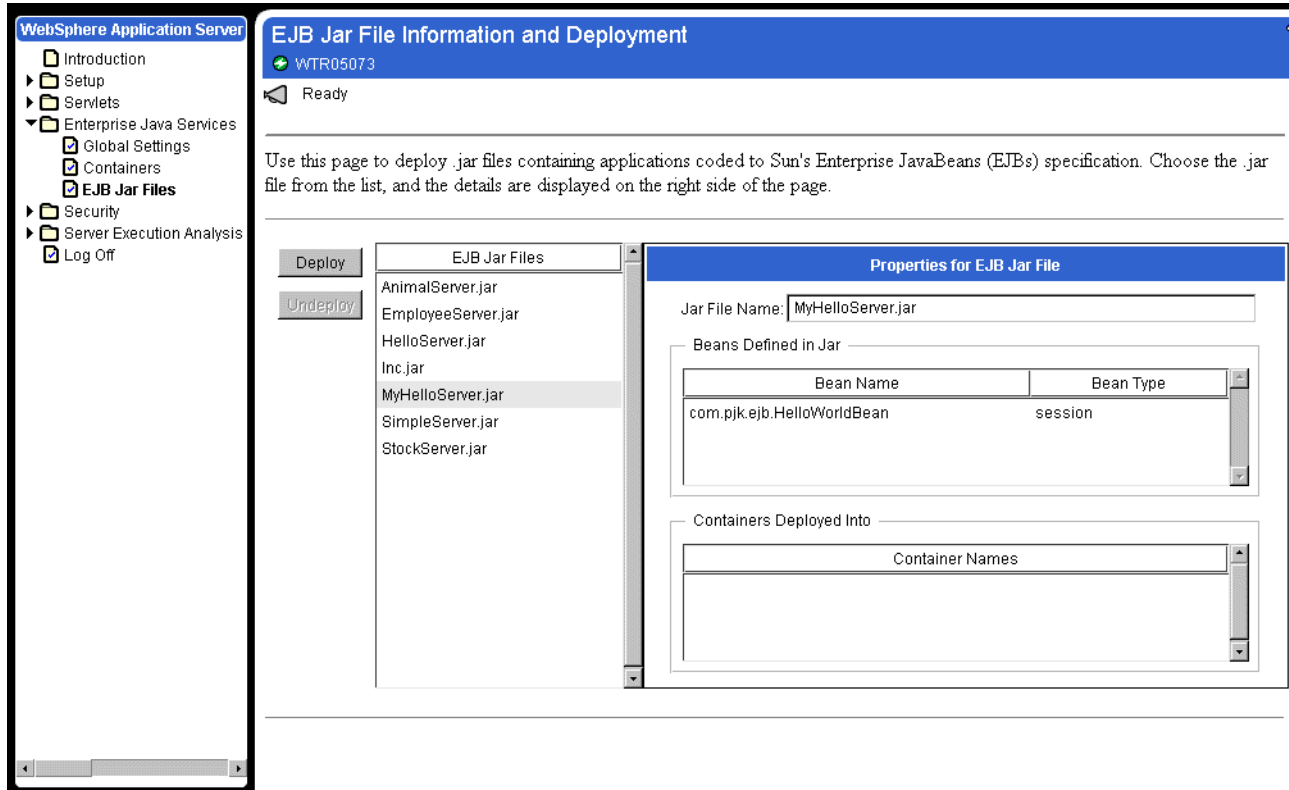


Figure 170. The EJB JAR Files Page

3. Select the JAR file for the EJB or EJBs that you want to deploy from the list and click **Deploy**.
4. In the container selection dialog box (see Figure 171) select the name of the container into which you wish to deploy. Make sure that you select a container type that is consistent with the EJBs contained within the JAR file. Deploy JAR files containing session EJBs into a session container, JAR files with entity EJBs into an entity container, and JAR files with both types of EJBs into both types of container.

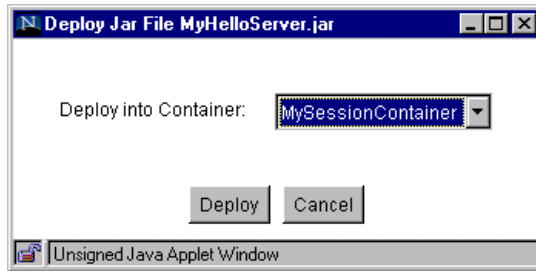


Figure 171. The Container Selection Dialog Box

5. If your JAR file already contains all of the stub and skeleton classes and the deployment descriptor you will see the Redeployment Selection dialog box (as shown in Figure 172). Choose either **Regenerate** to regenerate all of the stub and skeleton classes or **Redeploy Existing** to deploy the stub and skeleton classes already present in your JAR file. For more information on what classes are generated see 4.2.3.3, “WebSphere Generated Classes” on page 196.

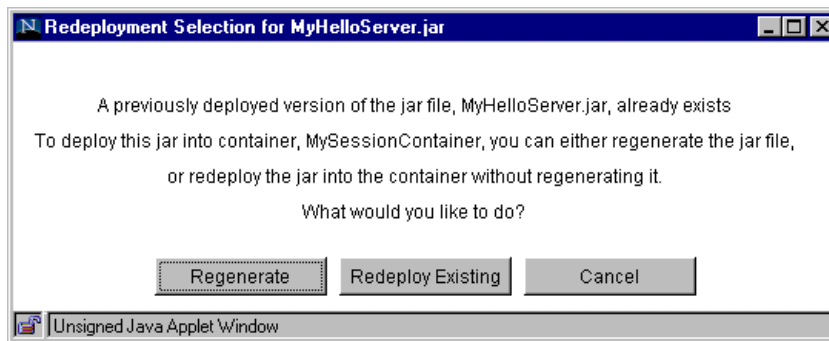


Figure 172. The Redeployment Selection Dialog Box

6. If all is successful you should see the dialog box shown in Figure 173 and the list of containers into which the bean has been deployed on the EJB JAR files page which should get updated to show your container (see Figure 174 on page 195).



Figure 173. The Successful Deployment Dialog Box

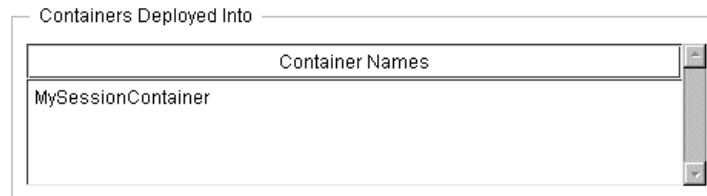


Figure 174. The EJB Deployment Container List

7. Stop and restart the EJS processes to activate the changes.

On Windows NT click **Start -> Settings -> Control Panel -> Services** and find the WebSphere Servlet Service. Stop this service by selecting it and click **Stop**. That will stop the EJS processes. Next stop and restart your Web server to restart EJS.

On AIX you will need to stop the Web server and then kill all of the WebSphere processes before restarting the Web server (see Figure 165 on page 187).

4.2.3.2 Removing an EJB from a Container

To remove or undeploy an EJB from a container, perform the following steps:

1. In the WebSphere administration interface select **Enterprise Java Services** and then **EJB JAR Files** (see Figure 170 on page 193).
2. Select the name of the JAR file you wish to remove and click **Undeploy**.
3. When the container selection dialog box comes up select the container you wish to remove the JAR file from and click **Undeploy**.



Figure 175. The Container Selection Dialog

4. If all goes well, you should see the successful completion dialog box shown in Figure 176 and the list of containers into which the JAR file is deployed (see Figure 174 on page 195), which should have been updated to show the change.



Figure 176. The Successful Undeployment Dialog Box

5. Stop and restart the EJS processes to activate the changes.

On Windows NT click **Start -> Settings -> Control Panel -> Services** and find the WebSphere Servlet Service. Stop this service by selecting it and clicking **Stop**. This will stop the EJS processes. Next stop and restart your Web server to restart EJS.

On AIX you will need to stop the Web server and then kill all of the WebSphere processes before restarting the Web server (see Figure 165 on page 187).

Note: The undeployment process removes the EJB from the container, but it does not delete the deployed EJBs JAR filename. For example, even after undeploying test.jar, you will still see the test.jar file in the \deployed EJBs directory.

4.2.3.3 WebSphere Generated Classes

When an EJB is in its deployed state in WebSphere, it needs a number of helper classes to allow client to server communication and to provide infrastructure for deployment. Some development tools can create some or all of these classes automatically and others do not. The EJB specification defines which classes must be in an EJB JAR file as follows:

- The class files for the bean or beans
- The class files for the home interfaces
- The class files for the remote interfaces
- The deployment descriptor for each EJB
- A manifest file describing the JAR file contents
- Any classes or resources referenced or needed by the EJB classes

Figure 177 shows an example of the contents for a simple EJB JAR file. Development tools that support EJBs will, at a minimum, support the generation and packaging of these classes. See your development tools' documentation for information on how to perform these tasks.

Name	Date	Time	Size	Ratio	Packed	Path
HelloWorld.class	05/07/99	10:55	244	22%	191	com\pjk\ejb\
HelloWorld.ser	05/07/99	10:55	1,258	48%	655	com\pjk\ejb\
HelloWorldBean.class	05/07/99	10:55	1,005	53%	477	com\pjk\ejb\
HelloWorldHome.class	05/07/99	10:55	285	29%	202	com\pjk\ejb\
manifest.mf	05/07/99	10:55	638	49%	326	meta-inf\

Figure 177. The HelloWorld EJB JAR File Before Deployment

When an EJB JAR file without all of the extra classes required by WebSphere is deployed, the deployment tools will generate a number of Java classes to

complete the deployment. Figure 178 shows the contents of the JAR file shown in Figure 177 after a successful deployment.

Name	Date	Time	Size	Ratio	Packed	Path
manifest.mf	05/07/99	11:04	2,180	63%	799	meta-INF\
HelloWorld.class	05/07/99	11:04	244	23%	188	com\pjik\ejb\
HelloWorld.ser	05/07/99	11:05	1,258	48%	652	com\pjik\ejb\
HelloWorldBean.class	05/07/99	11:05	1,005	53%	474	com\pjik\ejb\
HelloWorldHome.class	05/07/99	11:05	285	30%	200	com\pjik\ejb\
EJSHelloWorldHome.class	05/07/99	11:05	1,738	48%	901	com\pjik\ejb\
EJSRemoteHelloWorld.class	05/07/99	11:05	1,114	41%	661	com\pjik\ejb\
HelloWorldHelper.class	05/07/99	11:05	2,457	56%	1,078	com\pjik\ejb\
HelloWorldHolder.class	05/07/99	11:05	1,175	53%	548	com\pjik\ejb\
HelloWorldHomeHelper.class	05/07/99	11:05	2,505	57%	1,084	com\pjik\ejb\
HelloWorldHomeHolder.class	05/07/99	11:05	1,207	54%	552	com\pjik\ejb\
_HelloWorldHomeSkeleton.class	05/07/99	11:05	4,597	54%	2,130	com\pjik\ejb\
_HelloWorldHomeStub.class	05/07/99	11:05	5,671	54%	2,589	com\pjik\ejb\
_HelloWorldSkeleton.class	05/07/99	11:05	4,899	53%	2,321	com\pjik\ejb\
_HelloWorldStub.class	05/07/99	11:05	6,337	54%	2,917	com\pjik\ejb\

Figure 178. The HelloWorld EJB JAR File after Deployment

Most of these classes are used by the server only, with no direct reference by clients or modification by the server administrator. The only class that users need to reference is the <bean name>HomeHelper class. That class is generated for each EJB and has methods to assist clients in resolving references to names provided by the CORBA naming service in WebSphere. See 4.3.1, “Finding EJBs” on page 207 for more information.

The IBM VisualAge for Java EJB support, provided in the 2.0 release by the VAJ Enterprise Update fix, provides several options for exporting EJB JAR files. If you export an EJB JAR file, you only export the classes shown in Figure 177 on page 196. If, on the other hand, you export an EJS JAR file, all of the helper classes required for deployment in WebSphere are generated automatically and placed in the exported JAR file.

4.2.4 Working with Deployment Descriptors Using the Jet Tool

In order to specify how an EJB is deployed on a server, a special class called a deployment descriptor is used. This class specifies, for example, security constraints and transactional attributes. A deployment descriptor is an instance of the class `javax.ejb.deployment.DeploymentDescriptor`. A serialized version of this class must be included in the EJB JAR file prior to the deployment of the EJB to inform the server how to manage the EJB correctly. See the EJB specification at <ftp://ftp.javasoft.com/docs/ejb/ejb.1.0.pdf>, section 18 for more information on the deployment descriptor specification.

WebSphere provides a tool called Jet to assist in creating and editing deployment descriptors. Jet allows you to modify the deployment characteristics of EJBs prior to deployment and also to take JAR files containing EJBs and turn them into EJB JAR files through the addition of a deployment descriptor.

Another feature of Jet is that it allows the creation of XML documents containing the information required to create deployment descriptors for EJBs. Unfortunately we were not able to use these XML files to recreate the deployment descriptors due to bugs in the tool. The XML format used is different from the one used in the latest version of the EJB 1.1 specification available at the time of writing.

The Jet tool is not good at handling JAR files with more than one EJB present, which unfortunately includes most of the samples. If you need to work with a JAR file with more than one EJB it is recommended that you unpack the JAR file and either import the individual files into a development tool that can generate EJB deployment descriptors, such as VisualAge for Java, or assemble the files into separate JAR files for processing by Jet. This issue has been noted by the WebSphere developers and improvement is expected in Version 3.0.

4.2.4.1 Software Requirements

The Jet tool requires that you have the Java Foundation Classes (otherwise known as Swing) Version 1.0.3 available on your system. These classes are required for the user interface for the Jet tool. VisualAge for Java, if you have it installed, includes the swingall.jar file containing these classes in the <VAJava home>\lib\hpij\directory. Alternatively, they can be downloaded from <http://java.sun.com/products/jfc/index.html>.

Note: This is back-level from the latest release of these classes (1.11 at the time of writing).

4.2.4.2 Configuration

For Windows NT you will need to edit the file <Server Root>\samples\ejb\jet.bat and change the line:

```
set SWING_ROOT=.
```

to reflect the directory containing the swingall.jar file. For example:

```
set SWING_ROOT=D:\swing-1.0.3
```

For AIX there is no launch file provided, although it should not be difficult to edit jet.bat into a shell script if desired.

4.2.4.3 Using the Jet Tool Interactively

To use the Jet tool interactively you need to complete the configuration steps shown above. Once you have edited the batch file you can open a command prompt, change the directory to the <Server Root>\samples\ejb directory and type `jet` on the command line to run the tool. You should see the window shown in Figure 179:

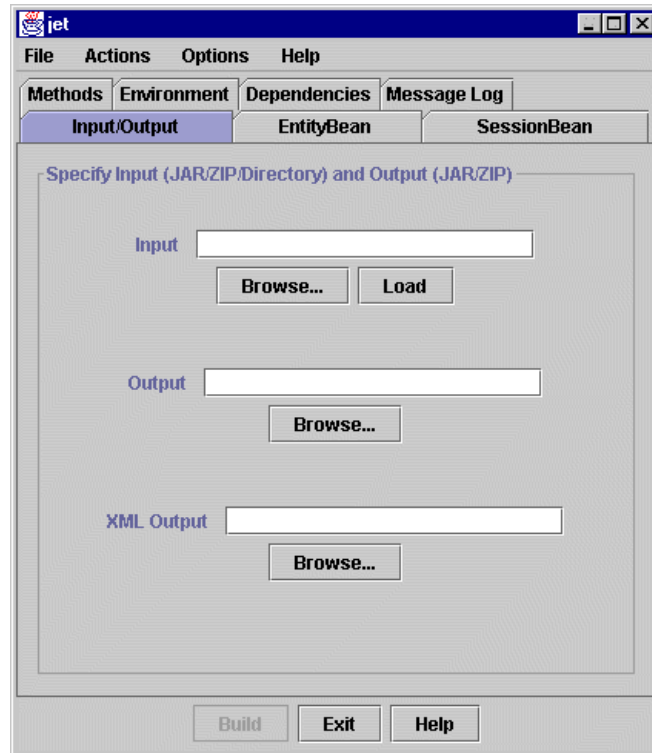


Figure 179. The Jet Initial Start-Up Screen

Note: All of the figures in this section will use the *Metal* look and feel. If you prefer you can use the options menu to set the look and feel to be either Windows or Motif. The functions in each case are the same.

Loading a Bean and Setting Paths Interactively

In order to load a bean, either click the **Browse** button located near the Input field on the front panel to locate a JAR file, or type the JAR file name and path into the Input field and click **Load**.

The documentation states that it is possible to type a directory name in this field and have the Jet tool load any EJB in that directory.

If there is more than one bean in the JAR file loaded, a dialog box may appear asking you with which bean in the JAR file you wish to work. There appears to be a problem with the Jet tool if there is more than one EJB in the same directory in the JAR file. It also appears that the tool incorrectly identifies classes that are not EJBs as being possible beans. It is recommended that EJB JAR files used with Jet either have only a single bean or have each EJB in a separate package directory.

Given the limitations of the Jet tool and the difficulties that VisualAge for Java has importing multiple EJB JAR files, it is recommended that each EJB JAR file contain only one EJB.

In the Output field, put the name of the JAR file that you want to build. This is the EJB JAR file that WebSphere will deploy. After the information is entered, the Build button should become active. This button causes Jet to create the EJB JAR file specified in the Output field.

Finally, Jet will output an XML version of the deployment descriptor for later batch processing. If this needs to be performed, enter the name of the XML file into the XML Output field before clicking **Build**.

It should be possible to load an EJB from a JAR file using the `-f` command line option and a prepared XML file, but we were unable to get this working due to null pointer exceptions in the Jet tool.

Working with Session Beans

If the bean you have loaded is a session bean then the SessionBean tab becomes active (see Figure 180). On this page it is possible to enter the parameters that govern the deployment of a session bean.

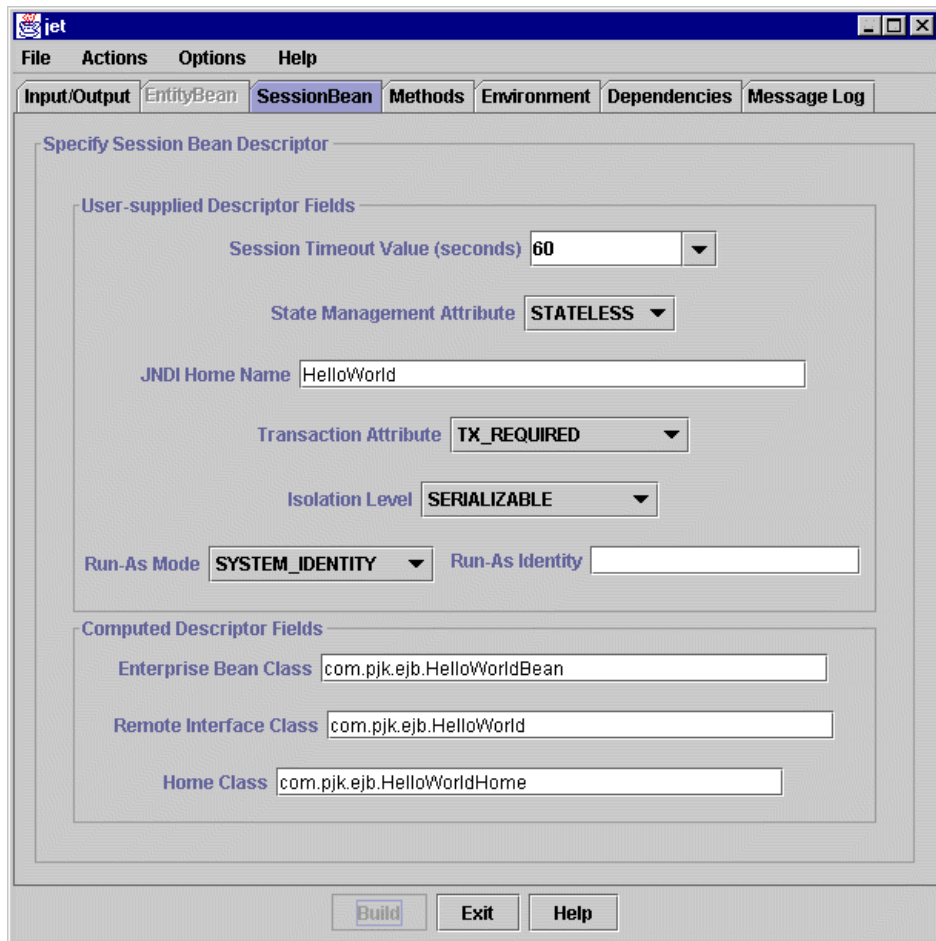


Figure 180. The Jet SessionBean Page

The fields on this page correspond to the properties of the `javax.ejb.deployment.SessionDescriptor` class, which inherits from the `javax.ejb.deployment.DeploymentDescriptor` class (see the EJB specification at <ftp://ftp.javasoft.com/docs/ejb/ejb.10.pdf>, section 18 for more information) according to Table 17. Note that the Run-As Identity value has no effect, because

security is not supported in WebSphere 2.02 (this is different from WebSphere's security).

Table 17. EJB SessionDescriptor Values

Field Name	Session Descriptor Property
Session Timeout Value (seconds)	sessionTimeout
State Management Attribute	stateManagementType
JNDI Home Name	beanHomeName
Transaction Attribute	Corresponds to the transactionAttribute property of an instance of the javax.ejb.deployment.ControlDescriptor class, with no method specified, added to the controlDescriptors property.
Isolation Level	Corresponds to the isolationLevel property of an instance of the javax.ejb.deployment.ControlDescriptor class, with no method specified, added to the controlDescriptors property.
Run-As Mode	Corresponds to the runAsMode property of an instance of the javax.ejb.deployment.ControlDescriptor class, with no method specified, added to the controlDescriptors property.
Run-As Identity	Corresponds to the runAsIdentity property of an instance of the javax.ejb.deployment.ControlDescriptor class, with no method specified, added to the controlDescriptors property. This value has no effect in WebSphere 2.02.
Enterprise Bean Class	enterpriseBeanClassName
Remote Interface Class	remoteInterfaceClassName
Home Class	homeInterfaceClassName

Working with Entity Beans

If the bean you have loaded is an entity bean, then the EntityBean tab becomes active (see Figure 181). On this page it is possible to enter the parameters that govern the deployment of an entity bean.

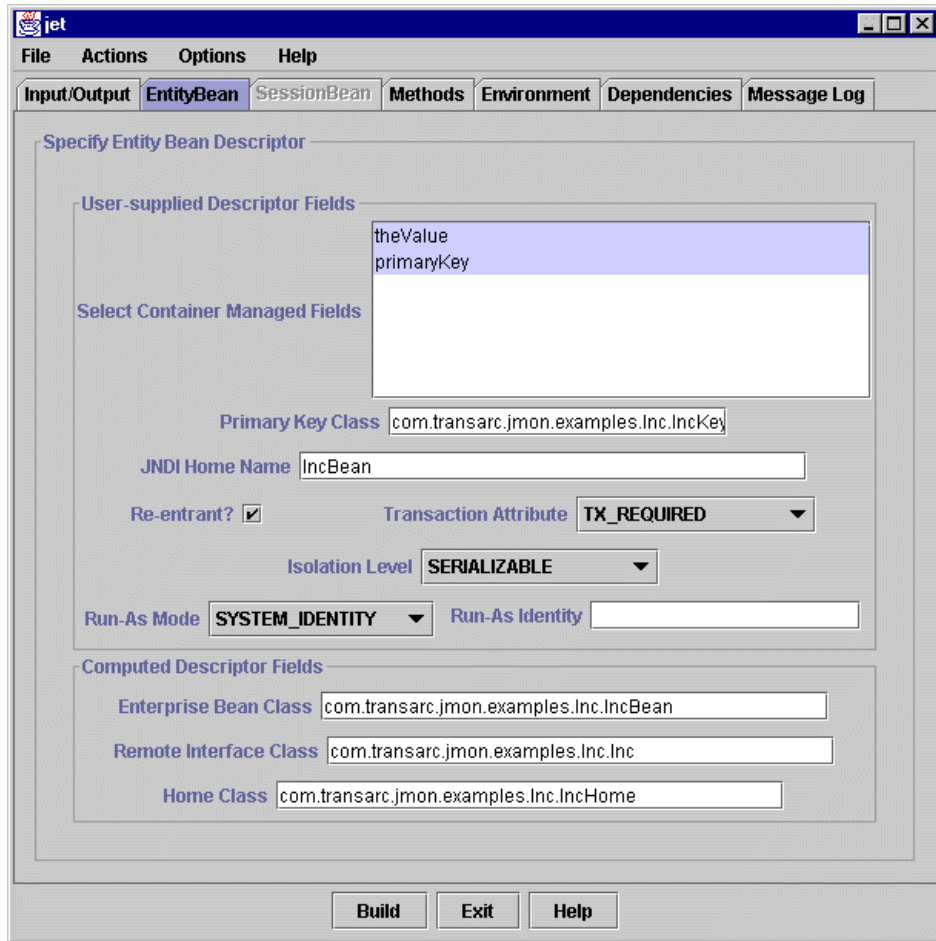


Figure 181. The Jet EntityBean Page

The fields on this page correspond to the properties of the `javax.ejb.deployment.EntityDescriptor` class, which inherits from the `javax.ejb.deployment.DeploymentDescriptor` class (see the EJB specification at <ftp://ftp.javasoft.com/docs/ejb/ejb.10.pdf>, Section 18 for more information) according to Table 18. Note that the Run-As Identity value has no effect, because EJB security is not supported in WebSphere 2.02 (this is different from WebSphere's security).

Table 18. EJB EntityDescriptor Values

Field Name	Session Descriptor Property
Select Container Managed Fields	containerManagedFields
Primary Key Class	primaryKeyClassName
JNDI Home Name	beanHomeName
Re-entrant?	reentrant
Transaction Attribute	Corresponds to the transactionAttribute property of an instance of the <code>javax.ejb.deployment.ControlDescriptor</code> class, with no method specified, added to the <code>controlDescriptors</code> property.

Field Name	Session Descriptor Property
Isolation Level	Corresponds to the isolationLevel property of an instance of the javax.ejb.deployment.ControlDescriptor class, with no method specified, added to the controlDescriptors property.
Run-As Mode	Corresponds to the runAsMode property of an instance of the javax.ejb.deployment.ControlDescriptor class, with no method specified, added to the controlDescriptors property.
Run-As Identity	Corresponds to the runAsIdentity property of an instance of the javax.ejb.deployment.ControlDescriptor class, with no method specified, added to the controlDescriptors property. This value has no effect in WebSphere 2.02.
Enterprise Bean Class	enterpriseBeanClassName
Remote Interface Class	remoteInterfaceClassName
Home Class	homeInterfaceClassName

Setting Method Properties

When you select the **Methods** tab, the method properties page is displayed (see Figure 182).

Note: While Jet allows you to enter these values, they are not supported by WebSphere 2.02. The values for each method will be taken from the default values set on the EntityBean and SessionBean pages.

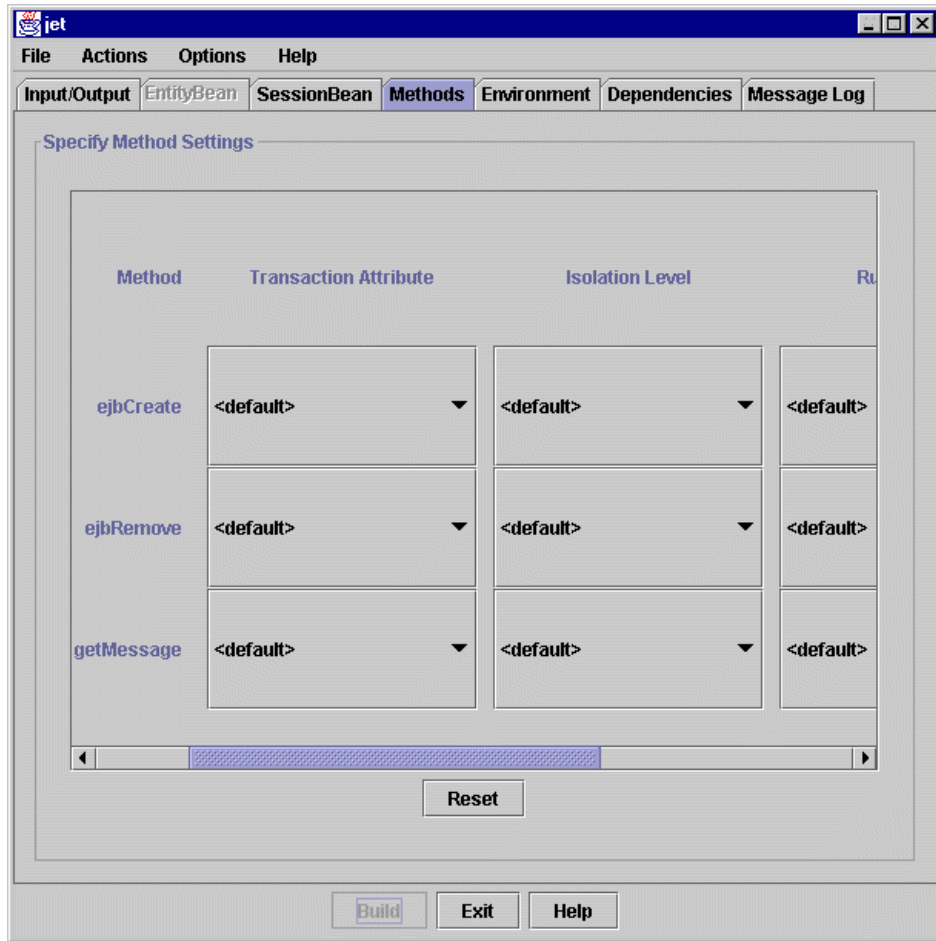


Figure 182. The Jet Method Properties Page

This page allows you to specify the deployment properties of the methods available in the EJB. Each row on the page corresponds to a method in the EJB and each column corresponds to a property value for an instance of the `javax.ejb.deployment.ControlDescriptor` class with the appropriate method specified on the constructor (see section 18 of the EJB specification at <ftp://ftp.javasoft.com/docs/ejb/ejb.10.pdf> for the specification of the `ControlDescriptor` class). This instance is added to the `DeploymentDescriptor` controlDescriptors property. The column headings correspond to the properties of the `javax.ejb.deployment.ControlDescriptor` class according to Table 19:

Table 19. EJB ControlDescriptorValues

Column Label	Control Descriptor Property
Transaction Attribute	transactionAttribute
Isolation Level	isolationLevel
Run-As Mode	runAsMode
Identity	runAsIdentity

Setting Environment Values

Selecting the Environment tab brings up the environment values page (see Figure 183).



Figure 183. The Jet Environment Values Page

This page allows you to specify environment values that correspond to `java.util.Properties` objects that are used as the value for the `environmentProperties` property of the `javax.ejb.deployment.DeploymentDescriptor` class (see section 18 of the EJB spec at <ftp://ftp.javasoft.com/docs/ejb/ejb.10.pdf> for the specification of the `DeploymentDescriptor` class). Each property is specified by a key/value pair separated by an = sign. Comments are prefixed with a :. The Insert From button brings up a file selection dialog box that allows you to select a text file to load into the field.

Setting Dependencies

Selecting the **Dependencies** tab brings up the dependencies page (see Figure 184).

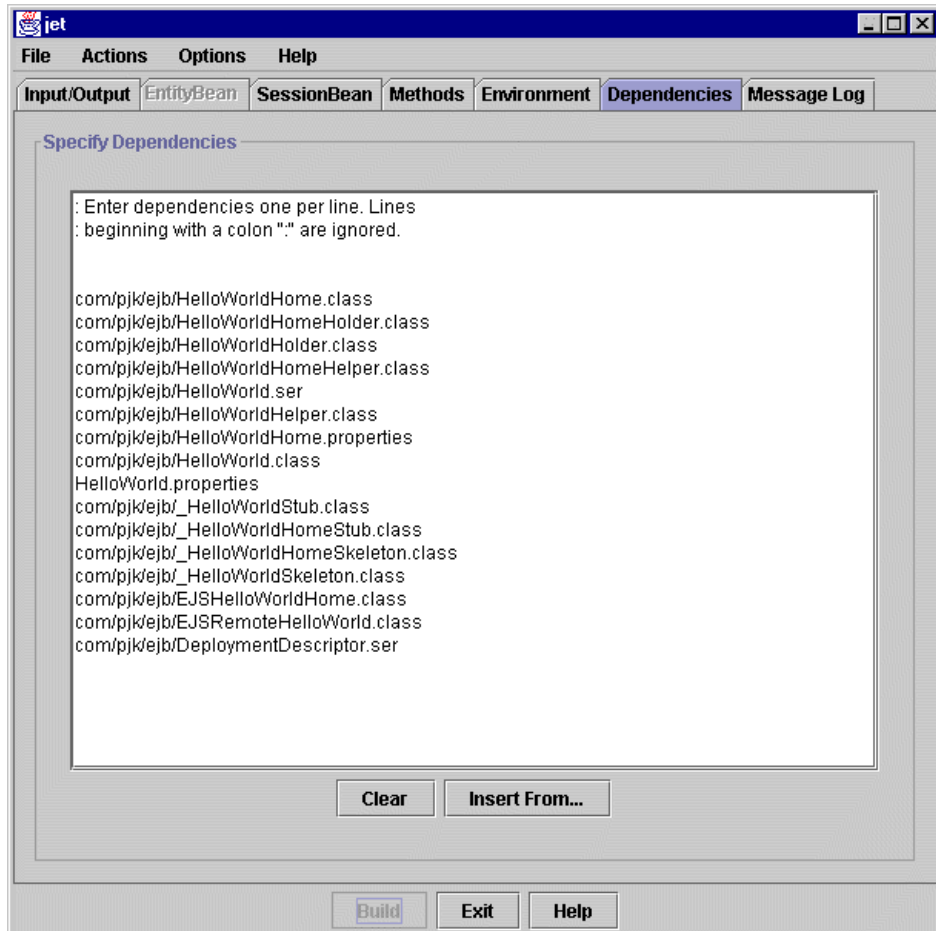


Figure 184. The Jet Dependencies Page

This page allows you to specify files in the JAR file on which the EJB depends. This information is included in the JAR file manifest file, along with a notation as to whether the file is included within the JAR file or is external to the JAR file. Comments are prefixed with a :. The Insert From button brings up a file selection dialog box that allows you to select a text file to load into the field.

Looking at Messages

When you select the **Message Log** tab, the messages page is displayed (see Figure 185).

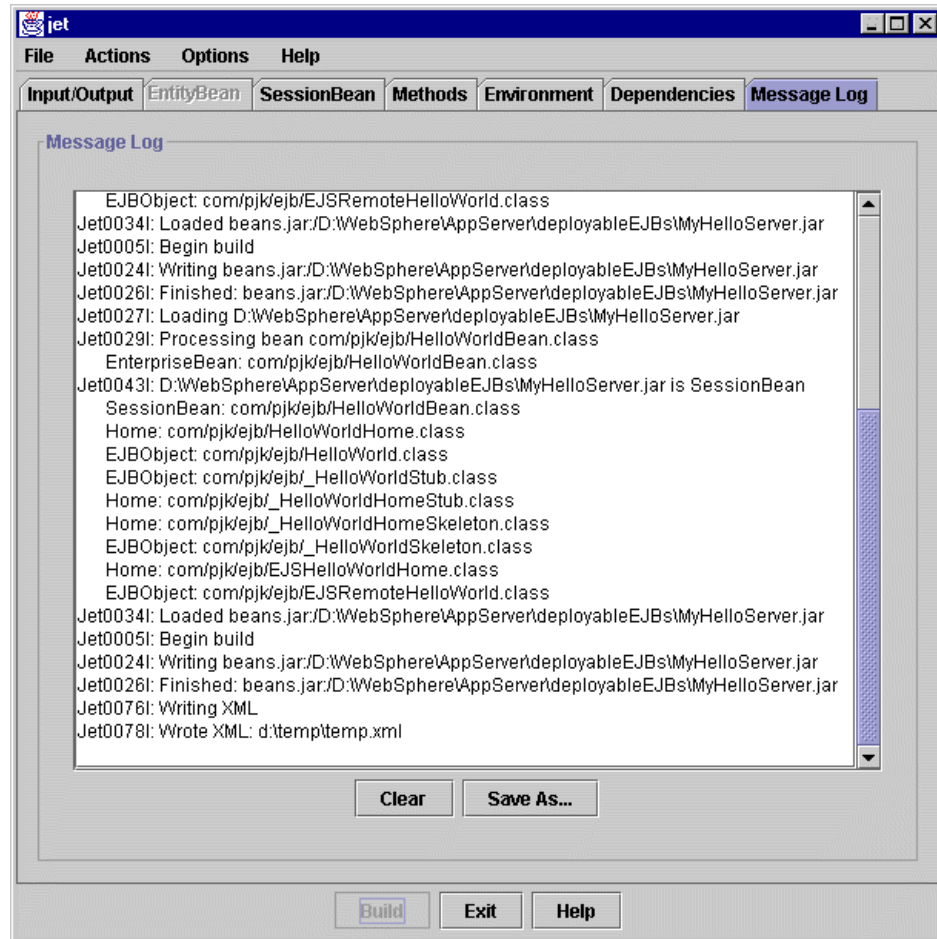


Figure 185. The Jet Messages Page

This page displays informational messages about the use of the Jet tool.

4.2.4.4 Using the Jet Tool in Batch Mode

It is possible to use XML files specifying deployment descriptor information to process EJB JAR files in batch mode. This is accomplished through the use of the `-x` command line option for Jet. Unfortunately we were not able to get this working, even with the WebSphere samples, due to null pointer errors when starting the tool.

4.3 Coding WebSphere EJB Clients

There are several considerations that must be taken into account when coding for clients to be able to access EJBs deployed on a WebSphere server as opposed to coding EJB clients in general.

4.3.1 Finding EJBs

In order for a client to be able to interact with an EJB, it must first be able to locate its home interface so that it can call either the create or find methods (see 1.7.4.2, “Steps in Using an EJB” on page 21). This is accomplished by means of a JNDI interface to a CORBA naming service provided by WebSphere. Covering JNDI or

CORBA in detail is beyond the scope of this book. However, there is a JNDI tutorial on Sun's Web site at <http://java.sun.com/products/jndi/tutorial/index.html> and an OMG CORBA for beginners page at <http://www.omg.org/corba/beginners.html>.

There are two possible methods that can be used by Java clients to access an EJB home instance for an EJB hosted on a WebSphere server:

1. Explicit instance location
2. Using the EJBHelper class

Note: See 4.3.1.3, "Using Client Transactions" on page 212 for some important considerations for using the EJS naming service and also using EJB client transactions.

4.3.1.1 Explicit Instance Location

The steps involved in explicitly locating a reference to an object's home interface are described below. You may wish to refer to Figure 186 on page 209 for an example taken from the Inc Client sample that is delivered with the WebSphere code:

1. Prepare a hashtable containing the properties necessary to create a JNDI initial context.

The constructor for the JNDI Initial Context requires a hashtable parameter. In this hashtable are stored the necessary pieces of information required to create an initial context. This initial context can then be used to look up the location of an EJB home instance. For the WebSphere naming service the following are the possible parameters:

- `javax.naming.Context.INITIAL_CONTEXT_FACTORY`

This constant is equal to the string "java.naming.factory.initial". This property identifies the actual name service that the EJB client must use. For WebSphere this value should be set to

```
"com.ibm.jndi.CosNaming.CNInitialContextFactory"
```

This value is required.

- `javax.naming.Context.PROVIDER_URL`

This constant is equal to the string "java.naming.provider.url". This property specifies the host name and port of the name server used by the EJB client. This is how a remote client specifies the address of a WebSphere server. The property value must have the following format:

```
"iiop://hostname:port"
```

where hostname is the IP address or hostname of the machine on which the name server runs and port is the port number on which the name server listens. If not specified, this property defaults to the local host and the port number specified on the EJS Global Properties page, usually 9019. For example, the property value

```
iiop://bankserver.mybank.com:9999
```

directs an EJB client to look for a name server on the host named bankserver.mybank.com listening on port 9999. The property value

```
iiop://bankserver.mybank.com
```

directs an EJB client to look for a name server on the host named bankserver.mybank.com at port number 9019. The property value

```
iiop:///
```

directs an EJB client to look for a name server on the local host listening on port 9019. This value is optional; its omission is identical to specifying iiop:///. Omitting this value is most commonly used for EJB clients or servlets accessing EJBs deployed on the same WebSphere server.

2. Create a JNDI Initial Context.

Call the `javax.naming.InitialContext` constructor with the properties file created in step 1 as an argument to obtain an initial context. If the client needs to use transactions, then it is safer to call the `com.ibm.ejs.client.EJClient.getInitialContext()` method instead, with the same arguments. This alternative method of getting the initial context sets up the transaction environment correctly for distributed transactions. See 4.3.1.3, “Using Client Transactions” on page 212 for further details.

3. Perform a JNDI lookup for the EJB's home instance using the initial context.

Use the `lookup` (`java.lang.String`) method on the `InitialContext` obtained in step 2 to get an object reference to a CORBA object representing the EJB home instance. The name of the home instance is specified in the EJB deployment descriptor packaged in the EJB JAR file.

4. Extract the home instance reference from the CORBA object returned

Use the `narrow` method on the `<bean name>HomeHelper` class to extract a reference to the home instance. The `<bean name>HomeHelper` class is generated during deployment (See 4.2.3.3, “WebSphere Generated Classes” on page 196).

5. Call methods on the home interface to create or reference the EJB.

For a session EJB, call one of the create methods on the home interface instance. For an entity EJB, call either one of the create methods or one of the `findXXXXXX` methods. This will give you a reference to an instance of the EJB.

```
Properties p = new Properties();
p.put(Context.INITIAL_CONTEXT_FACTORY,
      "com.ibm.jndi.CosNaming.CNInitialContextFactory");
InitialContext ic = new InitialContext(p);

Object tmpObj = ic.lookup("IncBean");
IncHome incHome = IncHomeHelper.narrow((org.omg.CORBA.Object)tmpObj);

Inc inc = null;

try {
    inc = incHome.create(new IncKey(primaryKey));
} catch (DuplicateKeyException ex) {
    inc = incHome.findByPrimaryKey(new IncKey(primaryKey));
}
```

Figure 186. Accessing an EJB Home Instance

4.3.1.2 Using EJB Helper Classes

There are two helper classes that are used by the EJB samples that are useful in locating EJBs:

1. <Server Root>\samples\ejb\com\transarc\jmon\examples\EJBHelper.java

Despite the location of this class, it is actually a member of the package `com.ibm.ejb.client.util` and is a utility class to help clients find the EJB Home instance of an EJB.

The aim of the EJB helper class is to hide the complexities of dealing with the differences between the different naming services that could be used by the server to locate the home instance of an EJB. It uses a standard naming convention for a properties file for each EJB to contain the information needed to deal with the naming service. Clients using the EJB helper class can then remain unchanged if the server needs to change the details of the naming service at any time. It also simplifies the code required by a client to find the home instance of an EJB.

Since WebSphere provides its own naming service, this is of minimal value for clients intending to access only WebSphere servers, other than simplifying the client code at the expense of creating an extra properties file. If, however, a client needs to remain portable across multiple heterogeneous EJB servers, then this class provides real benefit.

To illustrate the difference in the client code, consider two examples taken from the WAS EJB samples. In Figure 187 on page 210, you can see the number of steps required to obtain a reference to the EJB home instance for the Inc Bean sample. In contrast to this, in Figure 188 on page 210 you can see the number of steps required to obtain a home instance for the business Bean sample.

```
Properties p = new Properties();
p.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.jndi.CosNaming.CNInitialContextFactory");

p.put("java.naming.provider.url", "iiop://your.server.name:9020");

InitialContext ic = new InitialContext(p);
Object tmpObj = ic.lookup("IncBean");
IncHome incHome = IncHomeHelper.narrow((org.omg.CORBA.Object) tmpObj);
```

Figure 187. Get a Reference to an EJB Home Instance for the Inc Bean Sample

```
businessHome = (BusinessHome) EJBHelper.getEJBHome(
"com.ibm.ejs.samples.simple.BusinessHome" );
```

Figure 188. Get a Reference to an EJB Home Instance for the Business Bean Sample

The code in the EJB helper class also allows for the possibility that you may wish to define a subclass specific to a particular naming service and call that rather than the generic implementation provided. This allows the name lookup to incorporate additional functions, such as transforming a CORBA object

reference into a proper Java object reference, the procedure used in WebSphere.

In order to use the EJB helper class in your own applications you must do the following:

- Include an import statement in your client similar to the following:

```
import com.ibm.ejb.client.util.EJBHelper;
```
- Create a properties file for each EJB that will be accessed through the EJB helper.

The properties file should be named the same as the home interface of the EJB but with a .properties extension. It should be placed in the EJB JAR file as a resource in the same directory as the home interface file. Using VisualAge for Java, this means putting the properties file in a directory relative to <VA Java Home>\ide\Project Resources\\ and then using the dot-separated components of the package name as subdirectory names in the usual fashion. When the EJB JAR file is generated from VisualAge, be sure to include this file as a resource file. By doing this you will include the properties file in the JAR file with the correct path information.

For example, the properties file for the Business Bean sample is included in the JAR file under the path /com/ibm/ejs/samples/simple/BusinessHome.properties. If this file were included in a VisualAge for Java development environment it might be placed in the path D:\IBM\java\ide\Project Resources\Business Bean Sample\com\ibm\ejb\samples\simple\BusinessHome.properties.

The properties file should contain key/value pairs separated by equal signs. Table 20 on page 211 lists the possible keys and their values:

Table 20. Keys and Values for EJBHelper .properties Files

Key	Value	Mandatory
<Full package qualified name of the EJBHome>	Name of the EJB Home interface in the naming service. Typically this is just the name of the EJB Home.	Yes
com.ibm.ejb.client.util.EJBHelper	Fully qualified name of the helper class that subclasses EJBHelper to provide an implementation for a particular naming service. If this key is not present the default RMI lookup is used.	No
java.naming.factory.initial	Name of the factory class used to create the javax.naming.InitialContext for a particular naming service.	Yes
other parameters	Other parameters used in creating the javax.naming.InitialContext. These are typically naming service specific and may include things like user ID and password for the naming service.	No

Figure 189 on page 212 shows the .properties file from the business bean example. Note the use of the JBrokerHelper class that implements the

naming function for the WebSphere naming service. The source for this class can be found in <Server Root>\samples\ejb\com\transarc\jmon\examples\JBrokerHelper.java.

```
com.ibm.ejs.samples.simple.BusinessHome=BusinessHome
com.ibm.ejb.client.util.EJBHelper=com.ibm.ejb.client.util.CosNaming.JBrokerHelper
java.naming.factory.initial=com.ibm.jndi.CosNaming.CNInitialContextFactory
```

Figure 189. BusinessHome.properties

- Call (<YourHomeInterface>) EJBHelper.getEJBHome("<fully qualified home interface name>") in your code to get a reference to the home instance on the server. See Figure 188 on page 210 for an example from the Business Bean client.
2. <server root>\samples\ejb\com\transarc\jmon\examples\Utils.java

The utils class, from the package com.transarc.jmon.examples, includes a method, lookupUrl(String url), that performs a name lookup using the WebSphere naming service. This is equivalent to using the EJBHelper and the JBrokerHelper classes mentioned in the previous item. The advantage of using this class over the EJB helper class is that it requires less setup as properties files do not need to be created. The disadvantage is that it does not give much productivity benefit over coding the API calls manually and it is more prone to environment changes. Figure 190 on page 212 shows a code segment from the Stock Portfolio Manager sample that gives an example of how to use this class.

```
java.lang.Object o = Utils.lookupUrl(stockHomeName);
stockHome = StockHomeHelper.narrow((org.omg.CORBA.Object) o);
```

Figure 190. Using the Utils Class

4.3.1.3 Using Client Transactions

The EJB specification provides for the possibility that transactions may be used by EJB clients to control the execution of EJB applications. It specifies an interface, javax.jts.UserTransaction, for this purpose. It does not, however, specify how a client may get an instance of the current transaction context for an EJB for a server.

For a technical overview of how transactions work, see section 4 of the EJB Developer's Guide at <http://java.sun.com/products/ejb/developers-guide.pdf>.

WebSphere provides a means by which clients can obtain an instance of the current transaction through the org.omg.CosTransactions.Current interface. This interface is compatible with the CORBA transaction standard. This interface, while not directly compatible with the EJB standard, can be used to provide transactional support to EJB clients accessing EJBs deployed on WebSphere servers. If better interoperability with other EJB servers is desired, the WebSphere documentation provides some example code showing how the javax.jts.UserTransaction interface can be implemented using the org.omg.CosTransactions.Current interface. The code can be found at <http://<your server name>:9527/doc/what-is/ec007.html#transclient>.

For a client to obtain a reference to the current transaction it must call the static `com.ibm.ejs.client.EJClient.getCurrent()` method like this:

```
org.omg.CosTransactions.Current current =
com.ibm.ejs.client.EJClient.getCurrent();

current.begin();
// do transactional work
current.commit(false);
```

Note: In order to propagate transactions correctly across method calls, a client must either make sure that any calls to JNDI to obtain an initial context are made prior to obtaining the current transaction or that the initial context is obtained by calling the `com.ibm.ejs.client.EJClient.getInitialContext()` method rather than the new `javax.naming.InitialContext()` method. See 4.3.1.1, “Explicit Instance Location” on page 208 for more information on obtaining initial contexts.

4.3.2 Monitoring EJS

Enterprise Java Services provides two facilities to allow the monitoring of EJB execution.

4.3.2.1 The EJS Process Monitor

WebSphere provides an EJS process monitor to give status information on the three EJS processes. Access the process monitor by using the administration interface and clicking **Server Execution Analysis -> Monitors -> EJS Status** (see Figure 191):




EJS Process Name	State		Response Time
Location Service Demon	UP		:00.211
Persistent Name Service	UP		:00.210
EJS Runtime	UP		:00.190

Figure 191. Part of the EJS Status Monitor Page

Automatic update can be enabled by clicking the **Start View** button. Each of the three EJS processes is shown, along with the response time for a sample request to come back from each process.

4.3.2.2 EJS Trace Support

EJS trace support provides a facility to monitor the execution of each of the three processes making up EJS:

- The Persistent Naming Service
- The Location Service Daemon
- The EJS Server Process

Of the three, the EJS server process trace is the most useful and, not coincidentally, produces by far the most output.

To enable EJS tracing perform the following steps:

1. Ensure the global EJS Trace Enabled setting is set to yes (see 4.2.1, “Setting Up the Environment” on page 188 for details on how to do this).
2. In the administration interface click **Server Execution Analysis -> Trace** (see Figure 192 on page 214).
3. Select both the Tracer Name and LogFile check boxes for the following Tracers:
 - EJS_stdout
 - EJS_stderr
 - LSD_stdout
 - LSD_stderr
 - PNS_stdout
 - PNS_stderr

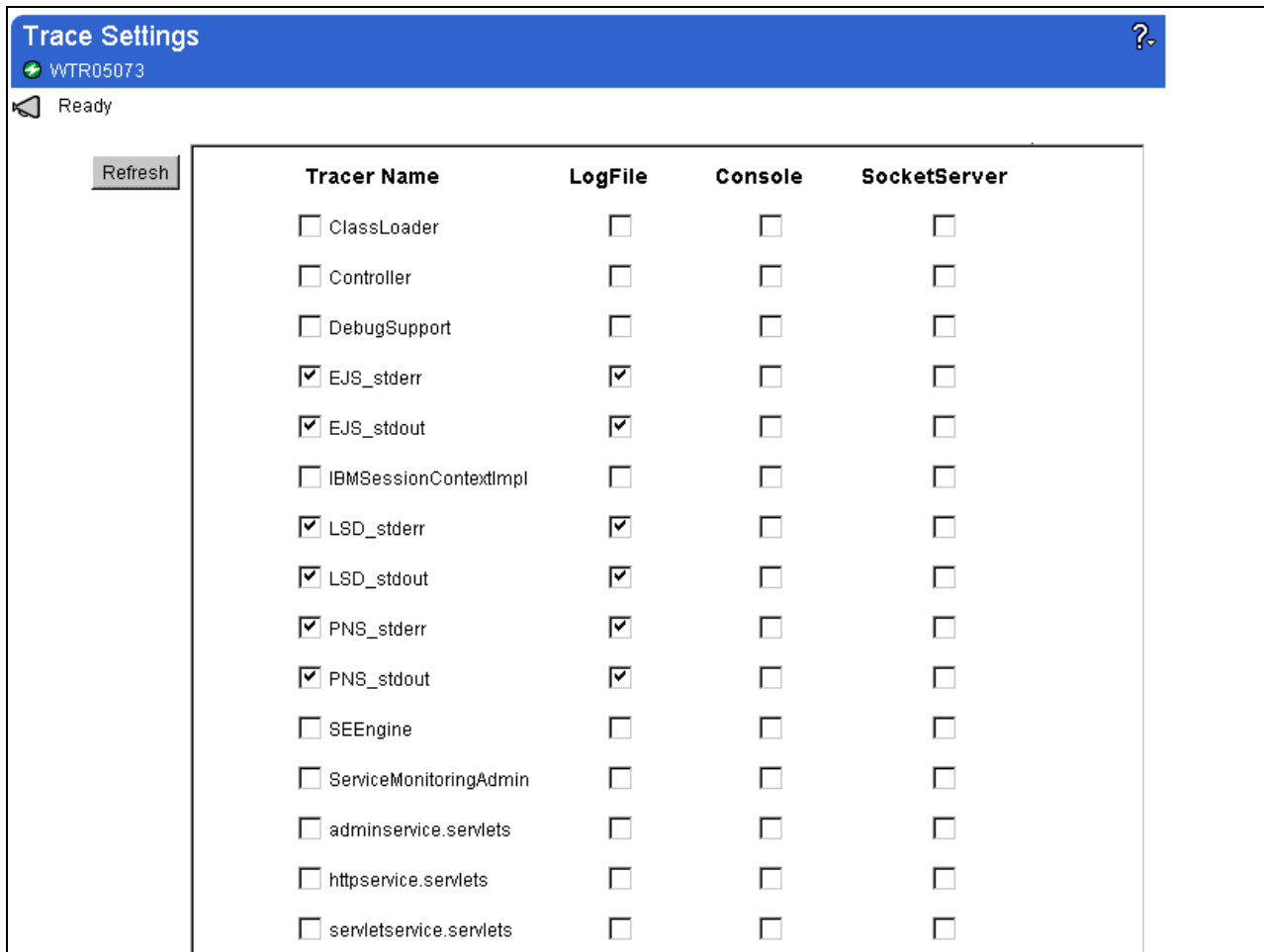


Figure 192. The Trace Settings Page

The changes to tracing should be effective immediately. After enabling these tracers you should begin to see EJS tracing information appearing in the <Server Root>/logs/websphere_trace.log file. Each log entry will be prefixed by a time and also the source of the tracer that produced it.

Figure 193 on page 215 shows an excerpt from the logs with output from all three EJS processes.

```

[5/18/99 3:10:31 PM EDT,EJS_stdout,1] 1099.461 00188def EJSContainer >
terminate
[5/18/99 3:10:31 PM EDT,EJS_stdout,1] 1099.461 00188def EJSContainer E
Container forcefully shutting down.
[5/18/99 3:10:31 PM EDT,EJS_stdout,1] 1099.481 00188de6 server >
ContainerManager$TerminationThread.run
[5/18/99 3:10:31 PM EDT,EJS_stdout,1] 1099.491 00188de6 EJSContainer >
terminate
[5/18/99 3:10:31 PM EDT,EJS_stdout,1] 1099.491 00188de6 EJSContainer E
Container forcefully shutting down.
[5/18/99 3:10:31 PM EDT,EJS_stdout,1] 1099.491 00188de6 EJSSessionCon E
EJSSessionContainer shutting down.
[5/18/99 3:10:31 PM EDT,EJS_stdout,1] 1099.501 00188de6 EJSContainer <
terminate
[5/18/99 3:10:31 PM EDT,EJS_stdout,1] 1099.501 00188de6 server <
ContainerManager$TerminationThread.run
[5/18/99 3:10:31 PM EDT,EJS_stdout,1] 1099.501 00188def EJSEntityCont E
TransarcEntityContainer shutting down.
[5/18/99 3:10:31 PM EDT,EJS_stdout,1] 1099.511 00188def EJSContainer <
terminate
[5/18/99 3:10:31 PM EDT,EJS_stdout,1] 1099.511 00188def server <
ContainerManager$TerminationThread.run
[5/18/99 3:10:31 PM EDT,EJS_stdout,1] 1099.511 00188e08 server <
ContainerManager.terminate
[5/18/99 3:10:31 PM EDT,EJS_stdout,1] 1099.521 00188e08 server <
ServerAdmin$TerminationThread.run
[5/18/99 3:21:28 PM EDT,EJS_stderr,1] Connection closed: Host =
wtr05073.itso.ral.ibm.com (port 2312)
[5/18/99 3:21:28 PM EDT,PNS_stderr,1] Connection closed: Host =
wtr05073.itso.ral.ibm.com (port 2304)
[5/18/99 3:21:28 PM EDT,PNS_stderr,1] Connection closed: Host =
wtr05073.itso.ral.ibm.com (port 2309)
[5/18/99 3:21:28 PM EDT,LSD_stderr,1] Connection closed: Host =
wtr05073.itso.ral.ibm.com (port 2307)

```

Figure 193. Excerpt from `websphere_trace.log` Showing EJS Tracing Information

4.4 Running the EJS Samples

There are a number of EJS samples shipped with WebSphere in order to provide illustrations of the use of the features of WebSphere EJS. Before running these samples it is necessary to perform a number of configuration steps to prepare the environment. After the configuration has been performed, the samples can be run using the EJS samples Web page, which is accessed via a link from the main WebSphere Samples Page or directly at:

`http://<your server name>/IBMWebAS/samples/ejs/index.html.`

Note: The samples have been designed to work with DB2. In order to configure them to work with other databases, coding changes to the source files would have to be made.

4.4.1 EJS Sample Configuration Steps

To run the EJB samples complete the following steps:

1. Create an EJS_SAMP database for the EJB samples.

In order to run the EJB samples, another database needs to be created in DB2 called EJS_SAMP. It is enough to create the database without any data. The samples will add the information later.

Follow the configuration instructions on your platform to create a new database with the name EJS_SAMP. The following command, when executed in a DB2 command line environment, is one way to accomplish this:

```
db2 create database EJS_SAMP
```

2. Create two new EJB containers.

In order to run the EJB samples you will have to create two new EJB containers, one for entity beans and one for session beans, and deploy EJBs into them. Follow the procedures detailed in 4.2.2.1, "Creating a Container" on page 190 to create two new containers with the values shown in Table 21:

Table 21. Parameter Values for Container Fields

Parameter	Value for Container 1	Value for Container 2
Container Name	Anything you choose	Anything you choose
Container Class	com.ibm.ejs.container.EJSEntityContainer	com.ibm.ejs.container.EJSSessionContainer
JDBC URL	jdbc:db2:ej_samp	n/a
EJB JAR Directory	deployedEJBs	deployedEJBs
Container User ID	Your DB2 logon ID	n/a
Container Password	Your DB2 Password	n/a

3. Deploy Sample EJB JAR files.

Follow the procedure shown in 4.2.3, "Deploying an EJB" on page 192 to deploy the sample EJB JAR files into the appropriate containers. There is already deployment information contained within the JAR files, so there is no need to regenerate it when asked.

Deploy the following JAR files into the entity container that you created in step 2:

- Animal.jar
- Inc.jar
- SimpleServer.jar
- StockServer.jar

Note that for each EJB in the JAR file, its type, entity or session is listed next to its name, allowing you to determine whether the JAR file should be deployed into an entity container or a session container or both.

Repeat the procedure to deploy the following files into the session container that you created in step 2:

- Animal.jar
- HelloServer.jar
- SimpleServer.jar
- StockServer.jar

4.4.2 Running the EJS Samples

The EJS samples can be accessed by selecting the **Enterprise JavaBeans** link from the samples page (<http://<your server name>/IBMWebAS/samples/>) or directly at <http://<your server name>/IBMWebAS/samples/ejs/index.html>. This brings up the Enterprise JavaBean Samples page (see Figure 194 on page 217) which has links to run the different EJB samples.

1... Hello.....
This is your standard 'Hello World' program written as a stateless session Bean. It's familiar, it's very simple, and it's a good place to start. [Say Hello!](#)

.... 2... Inc.....
Inc is short for increment. This sample increments a number stored in the database by adding the value 5. It illustrates a container-managed entity Bean. The stored number represents the inventory of banana cream pies. The application counts how many pies it is adding to the inventory and displays the new total. [Add five pies](#)

..... 3... Phone.....
Put your company's directory on your intranet. The Phone sample is a simple phone book application, based on an existing DB2 table of employee information. It illustrates a container-managed persistence entity Bean. When prompted, try it out by typing a name like *Smith* or *S* in the entry field. [Find a phone number](#)

..... 4... Animal Game.....
The Animal game is a version of the Twenty Questions game. It has animals and information about them stored in a database. You think of an animal, the Animal game asks you Yes/No questions, and then tries to guess the animal. It employs a knowledge base that grows each time you play the game. This application is implemented as a container-managed entity Bean that is read and updated by a stateful session Bean. [Play the Animal game](#)

..... 5... Stock Portfolio Manager.....
Buy and sell stock. This sample combines existing Stock and Portfolio EJBs into one application. You provide the stock symbol and a quantity. The Stock Portfolio Manager gets the current price from the Internet, displays the transaction total, and maintains your portfolio.
[Manage my stock](#)

Note: If the Application Server is running behind a firewall, the Stock servlet will need to get past it to the Internet. This usually means a socksified TCP/IP stack, which is not possible on AIX and Sun systems.

..... 6... Simple.....
Don't be fooled by its name. The Simple sample is not so simple. It has a session Bean, a container-managed entity Bean, and a Bean-managed entity Bean. And, it is the only sample that does Bean-managed persistence! Here's how it works: The Bean-managed entity Bean creates its own table to store its values, the container-managed entity Bean tells the container which fields to store, and, when deployed, creates the table in the database. [Start Simple](#)

Figure 194. Part of the EJB Samples Page

Each of the samples is described on the page and provides a link to execute the sample. The source code for each of the samples can be found in `<server root>\samples\ejb\com\ibm\ejb\samples\<sample name>`. Additional notes on the samples are provided below.

Hello

This is a very basic sample that uses a stateless session bean. See <http://<your server name>:9527/doc/what.is/ec010.html#HDRHELLO> for more details on the workings of this sample.

Inc

This is a basic entity bean sample that stores a simple counter in the database. Try executing the sample from different machines and see how the state is maintained across multiple clients.

See <http://<your server name>:9527/doc/whatis/ec010.html#HDRINC> for more details on this sample.

Phone

This sample uses an entity bean with container managed persistence to wrapper the employee data in the DB2 sample database in order to provide phone lookup services.

See <http://<your server name>:9527/doc/whatis/ec010.html#HDRPHONE> for more details about this sample.

Animal

This is an implementation of the classic beginners artificial intelligence program Animals. It uses a stateful session bean to run the game and an entity bean to maintain the knowledge tree. Figure 195 shows the message when the game correctly guesses your animal.

See <http://<your server name>:9527/doc/whatis/ec010.html#HDRANIMAL> for more details about this sample.

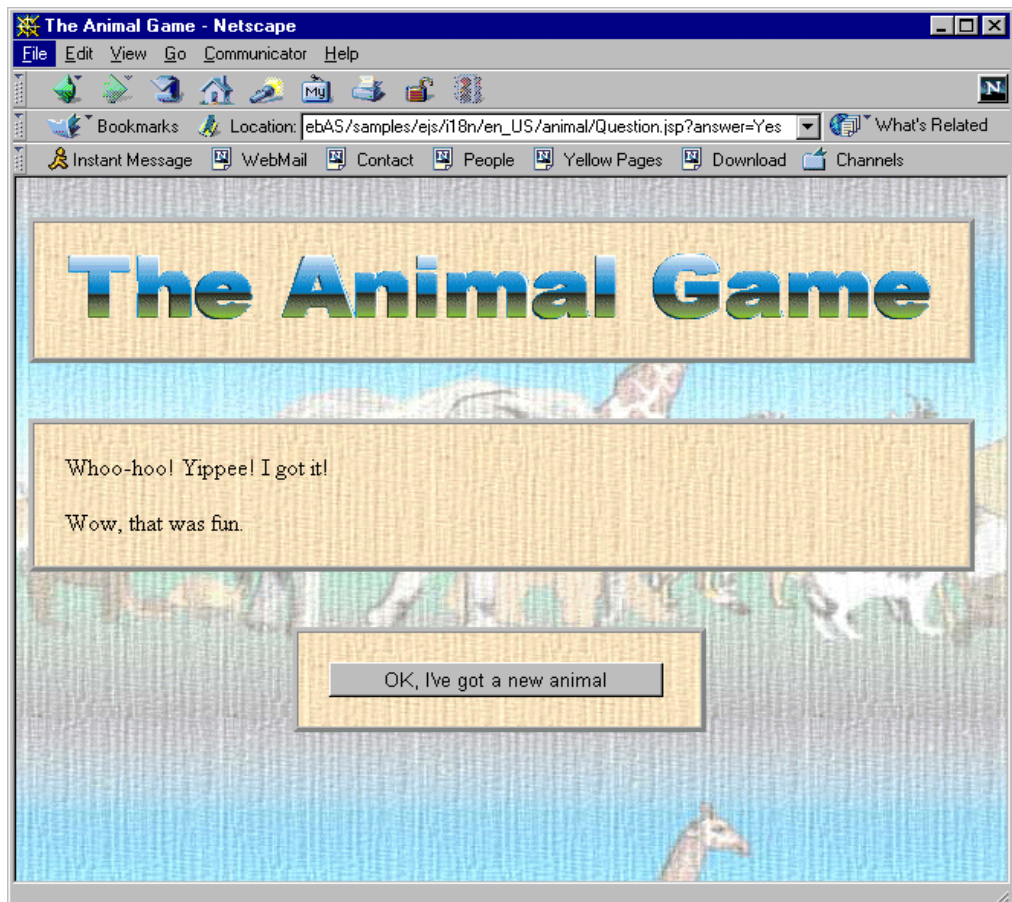


Figure 195. Playing the Animal Game

Stock

This sample uses real stock price data taken from the Internet to store information on an imaginary stock portfolio. The sample uses an entity bean to maintain the portfolio information and a session bean to perform the buy and sell transactions. This sample also uses the EJB transaction support.

Note: In order to run this sample your machine will have to be able to access the Internet directly. If you are running behind a firewall you will need to "SOCKSify" your TCP/IP stack using a SOCKS client. We were able to get the stock sample to work SOCKSified with little trouble from behind our firewall.

Figure 196 on page 220 shows the result of investing large amounts of imaginary money in some technology stocks. To help you get started in testing this sample Table 22 shows the stock symbols and corresponding companies for the four stocks shown in Figure 196:

Table 22. Ticker Symbols for Common Technology Stocks

Ticker Symbol	Company
IBM	International Business Machines Corp.
SUNW	Sun Microsystems
INTC	Intel Corporation
MSFT	Microsoft Corporation

See <http://<your server name>:9527/doc/what is/ec010.html#HDRSTOCK> for more information about this sample.

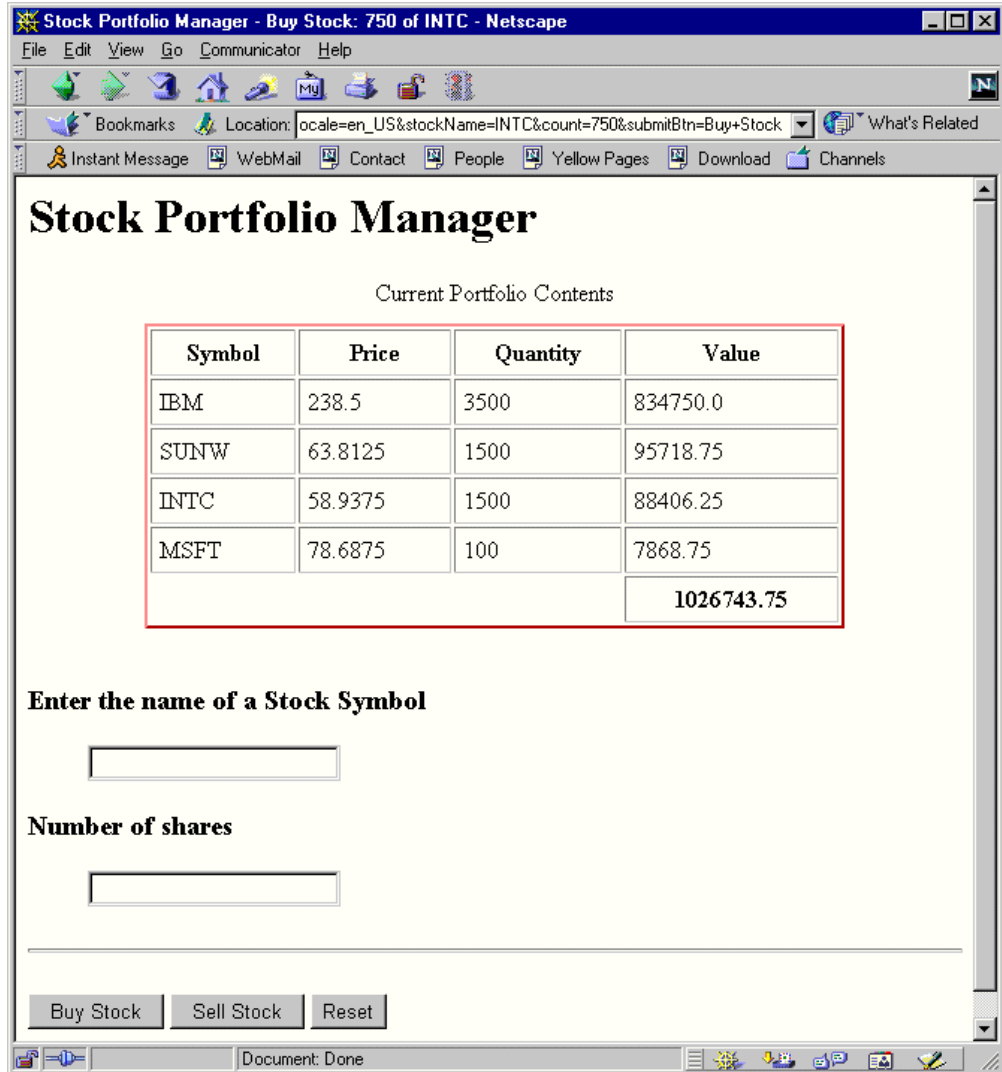


Figure 196. The Stock EJB Sample

Simple

The three simple samples, Business Bean, Customer Bean and Cart Bean, implement a business record, a customer record and a shopping cart respectively. With the business and customer beans, make sure you add and delete records from the end of the list only (for example, use only sequential numbers); otherwise, the JSP that displays them will not find them. The logic reads sequentially until it cannot find the next number in sequence and then stops. If you look in the <server root>\logs\jvm_stdout.log file you can see the error messages as the bean attempts to read past the end of the data and fails to find the next number in sequence.

See <http://<your server name>:9527/doc/whatis/ec010.html#HDRSIMPLE> for more details about these samples.

Chapter 5. Designing Applications for WebSphere

Web applications are becoming more and more important in the day-to-day running of businesses. Along with this new focus comes significant complexity in the design of the applications. There are several reasons that they are becoming more complex. The need for scalability, manageability, performance, persistence and reliability are driving new requirements to be implemented faster and faster. WebSphere provides substantial tools to help meet these requirements.

In this chapter we also discuss details about session management. What is session management and the reason why session management is important.

5.1 Session Management

A session is a connection between a client and a server where information is exchanged. This section explains the problem in Web applications caused by the stateless HTTP feature and provides hints and tips on how to handle those sessions using the HTTP protocol.

5.1.1 Maintaining HTTP Sessions

The HTTP protocol is one of the most important protocols used in Web applications. The HTTP protocol is stateless, that is, each request arrives carrying only its request context. For example, if someone orders two books in two different requests, the order gets accepted as two orders from different users as shown in Figure 197:

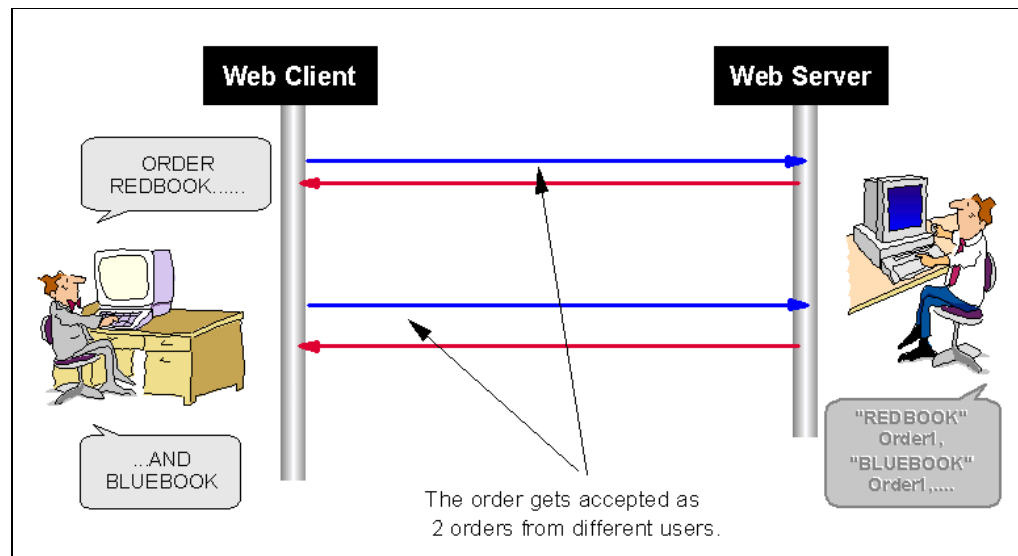


Figure 197. Stateless HTTP Session

5.1.1.1 Using Cookies to Solve Stateless HTTP Sessions

Cookie technology was developed by Netscape Communications. Cookies are named and have a single value. Cookies enable you to maintain context across requests using the following steps:

1. A cookie gets generated in the Web server and then it gets sent to the Web client.
2. The browser receives the data and maintains the information in memory or it stores the data in a text file on the hard disk. An example of the cookie file follows:

```
# Netscape HTTP Cookie File
# http://www.netscape.com/newsref/std/cookie_spec.html
# This is a generated file! Do not edit.

secure.webconnect.netFALSE/cgi-binFALSE12341173080873
990602114622957510011001
nemesis.makuhari.japan.ibm.comFALSE/~wakakoFALSE1924991999Times2
nemesis.makuhari.japan.ibm.comFALSE/~wakakoFALSE1924991999Date
1999/06/02%2011%3A53%3A31
.amazon.comTRUE/FALSE928828709session-id-time928828800
.amazon.comTRUE/FALSE928828709session-id002-1523943-4847406
.amazon.comTRUE/FALSE2082787110x-mainEm520rtC2lzcimMpKai4WG?2F6Tmp1rM
.amazon.comTRUE/FALSE2082787110
:
```

Figure 198. Cookie Information

3. The Web server gets the data from the client machine.

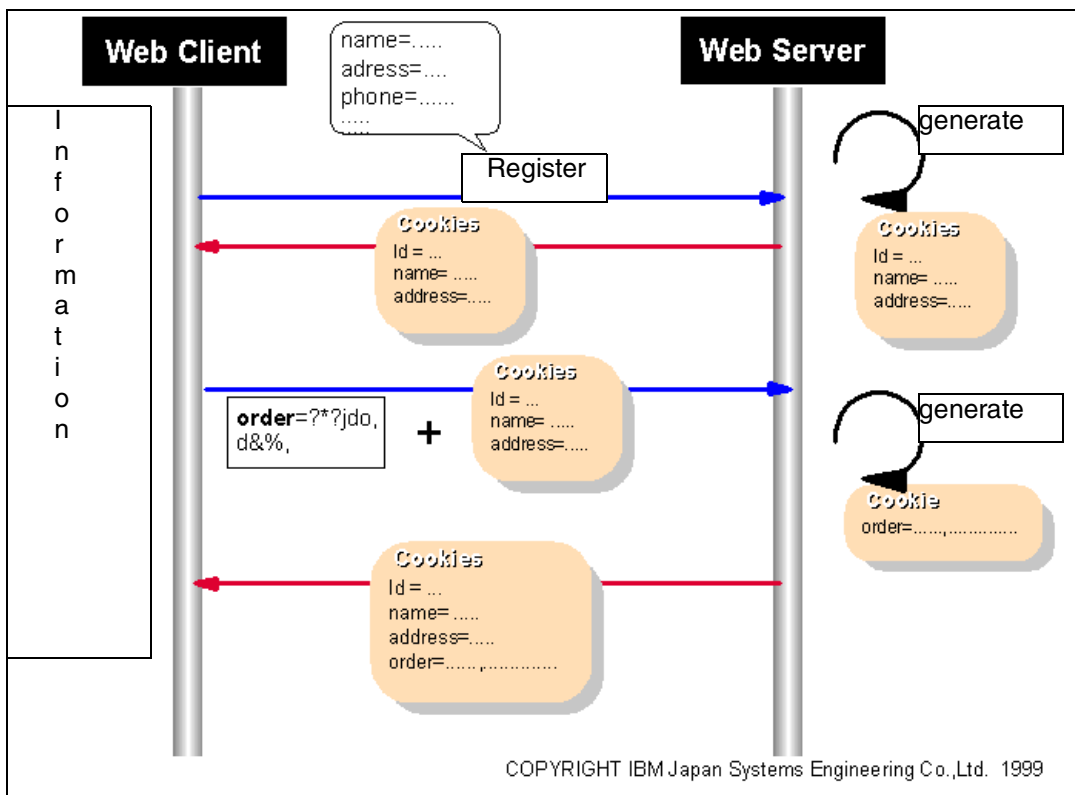


Figure 199. All Information Is Sent in Each Request

Cookies provide a convenient technology to logically handle stateful sessions, but they have some weak points:

- The information saved in a cookie is easily seen.

It is possible to trace information on most networks or read it from a hard disk since the information is typically not encrypted. Therefore, secure information, such as credit card numbers or personal information, shouldn't be stored in cookies.

- All information is sent in each request.

If you set your personal information, such as name, address and phone number, the information is sent as a cookie in each request while you are doing your online shopping (as shown in Figure 199 on page 222). This is not secure and it also causes increased HTTP traffic to flow. It usually doesn't affect network performance too much (since the amount of data is typically small), but it is not a sophisticated application design.

For more information about cookies, see

http://www.netscape.com/newsref/std/cookie_spec.html.

5.1.1.2 HttpSession Object

HttpSession is another technological implementation that is used to manage sessions on the stateless HTTP protocol. It is part of the Java Servlet API and it is accessible in the Java Servlet development kit.

- The HttpSession object is the Java programming object that has a dictionary-like interface to store user-defined data.

You can store data in the HttpSession object with any keyword (name) and look up the data with that keyword.

- Each of the session objects has its own ID.
- HttpSession objects are related to the client via the session ID.

The Session Tracker uses the session ID to find the user's session object.

- It uses the cookie to carry the session ID that is associated with the client.
- You can also use URL rewriting (see 5.1.3.3, "Session Object Using URL Rewriting" on page 239 for more details).
- The session object is not sent between the Web client and the server.

Only the ID is passed back and forth between the Web client and the Web server (Figure 200 on page 224). This means the session object is suitable to store the user's secure data.

- The data stored in the HttpSession object is shared across requests from the same machine.

Therefore, the HttpSession object is suitable to store individual data from the client.

- The HttpSession object is not persistent data. It is deleted after a timeout period, or by the invalidate() method. In order to store data persistently, WebSphere provides you with a UserProfile (see 5.2, "User Profiles" on page 253). The HttpSession description can be found at

http://java.sun.com/products/servlet/2.1/api/javax.servlet.http.HttpSession.html#_top_.

The HttpSession interface is: *javax.servlet.http.HttpSession*.

More information on session tracking can be found at
<http://java.sun.com/docs/books/tutorial/servlets/client-state/session-tracking.html>.

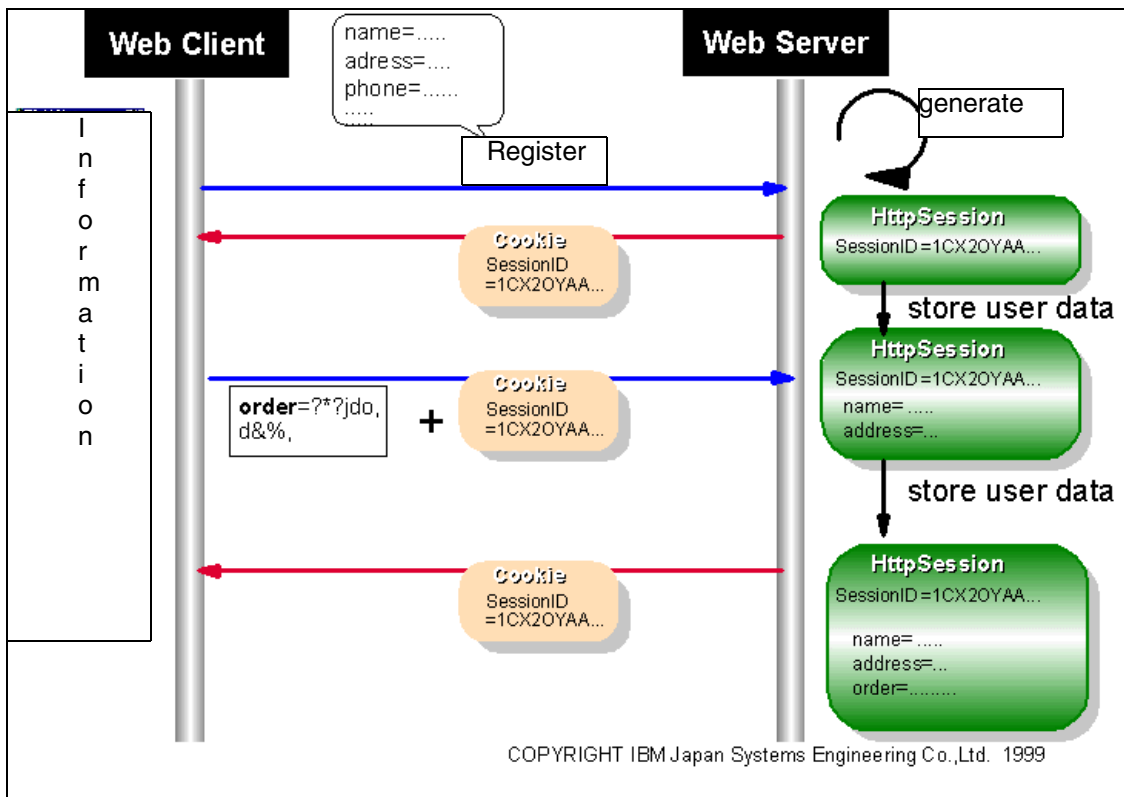


Figure 200. Only Session ID Is Passed Between Web Client and Server

5.1.1.3 How the HttpSession Object Is Created

The HttpSession object is created on the WebSphere server with the first request from a user. To get an instance of HttpSession, use the following method:

```
public abstract HttpSession getSession(boolean create)
```

This method gets the current valid session information associated with this request. It does that if the value of *create* is false or, if necessary, it creates a new session for the request, if the value of *create* is true.

If the HTTP request is authorized, the authorized name is set to HttpSessionObject user name; if not, an anonymous name is set, as shown in Figure 201 on page 226.

5.1.1.4 When Is the HttpSession Object Deleted?

The HttpSession object is deleted under the following two conditions:

1. TimeOut

After a time period defined by the Application Server Manager (the default setting is 30 minutes) the HttpSession object is deleted.

2. When a specific method is used

You can delete the HttpSession object using the HttpSession.invalidate() method.

5.1.2 Session Tracking in the WebSphere Application Server

The WebSphere Application Server provides you with convenient functions for session tracking. In the WebSphere Application Server environment, you can:

- Monitor active sessions and set up the session environment from the GUI
- Control the life cycle of HttpSession/IBMSessionData objects
- Share session objects among multiple WebSphere Application Servers with technology called session clustering
- Make your data persistent using UserProfile ()
- Easily develop applications extending existing classes

5.1.2.1 IBMSessionData

The IBMSessionData class is the IBM extension to the implementation of HttpSession. The IBMSessionData inherits the functions from HttpSession and provides the following additional information for the user's convenience:

- Username
If the Http request is authorized, the authorized name is set to HttpSession object username. If it is not authorized, then the name is set to anonymous. In IBMSessionData, Username is also defined using the setUsername() method.
- UserPosition
The user's position within the Web site. For example:
/servlet/IbmSessionSample
- Referrer
The URL the user was visiting before this session was created.
- Message
Any user-defined message.

5.1.2.2 Monitoring HttpSession/IBMSessionData Object

You can monitor HttpSession/IBMSessionData objects from the Application Server Manager.

You can see this monitor by clicking **Server Execution Analysis -> Monitors -> Active Sessions** as shown in Figure 201 on page 226.

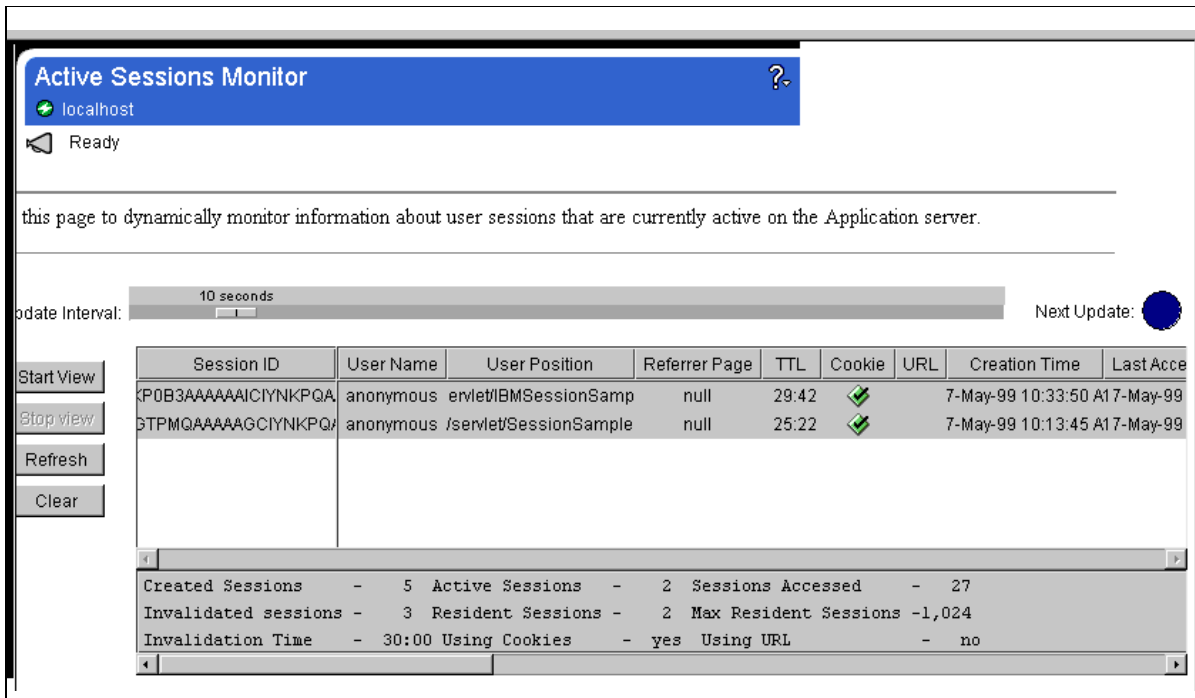


Figure 201. Active Sessions Monitor

The middle of this monitor shows each session object that exists on the server. A summary of each of the session's objects fields is shown in Table 23:

Table 23. Summary of the Field about Session Object

Field	Comment
Session ID	The unique ID that identifies a particular session.
UserName	The user name of the user logged in to the session.
UserPosition	The servlet or Java Server Pages (JSP) the user last visited during this user session. Usually, this value is the return value from a request.getRequestURL() method call. However, the value can be overridden using the setUserPosition() method.
Referrer Page	The URL of the Web page on which the user began the current session. If the current session began by directly invoking the servlet URL, this field is null.
TTL	Time to Live displays (in hh:mm:ss format) how long the current session will remain active.
Cookie	Whether the session ID was set with a cookie for this session (check mark if true).
URL	Whether the session ID was set via URL rewriting for this session (check mark if true).
Creation Time	When the session was created.
Last Accessed Time	The last time the session was accessed.

Status Information

The bottom of the page displays status information about active sessions.

Table 24. Summary Information about Active Sessions Status Field

Status Field	Comment
Created Sessions	Sessions created since the time the application server was last started.
Invalidated Sessions	Sessions invalidated since the time the application server was last started.
Invalidation Time	The amount of time a session is allowed to go unused before it is no longer validated, as configured in the Invalidate Time field on the Session Tracking page.
Active Sessions	The current number of active sessions.
Resident Sessions	The number of resident sessions in memory at this time.
Using Cookies	Whether cookies are being used to track Session IDs, as configured on the Session Tracking page.
Sessions Accessed	The number of sessions accessed since the time the application server last started.
Max Resident Session	The maximum number of sessions allowed to remain in memory at one time, as configured in the Maximum Residents field on the Session Tracking page.
Using URL	Whether URL rewriting is being used, as configured on the Session Tracking page.

5.1.2.3 Session Tracking Setting

To see how session tracking is set up, click **Setup -> Session Tracking** from the Application Server Manager.

There are five tabs: Enable, Cookies, Intervals, Persistence, and Host.

We discuss only the Enable and Cookies tabs in this section. For the Intervals tab, see 5.1.2.4, “Swapping” on page 230. For information on the Persistence tab, see 5.1.2.5, “Persistence” on page 231. The Host tab is explained in 5.1.4.1, “Session Management Modes” on page 248.

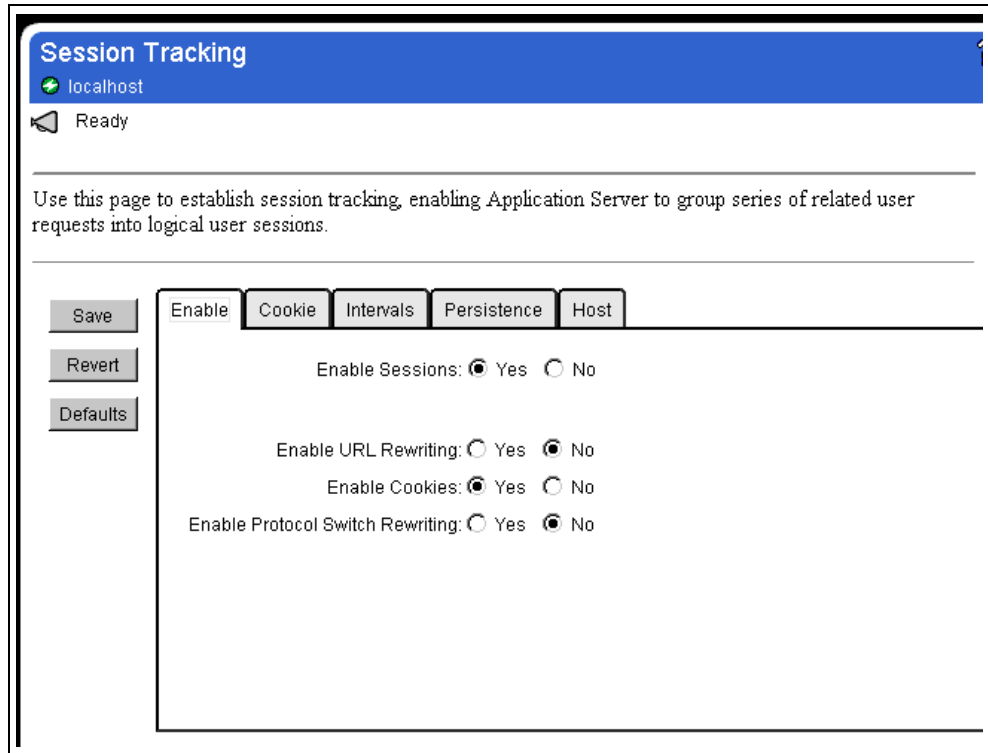


Figure 202. Enable Tab from Setup -> Session Tracking

Table 25. Fields on Enable Tab

Field	Comment
Enable Sessions	Whether session tracking is enabled, meaning the session-related methods for the request and response objects will be functional. Default: Yes
Enable URL Rewriting	Whether session tracking uses rewritten URLs to carry the session IDs. If it is enabled, the Session Tracker recognizes session IDs that arrive in the URL and, if necessary, rewrites the URL to send the session IDs. Default: No
Enable Cookies	Whether session tracking uses cookies to carry the session IDs. If Yes, session tracking recognizes session IDs that arrive as cookies and tries to use cookies as a means for sending the session IDs. Default: Yes
Enable Protocol Switch Rewriting	Whether the session ID is added to URLs when the URL requires a switch from HTTP to HTTPS or from HTTPS to HTTP. Whether the session is passed to different protocols. Default: No

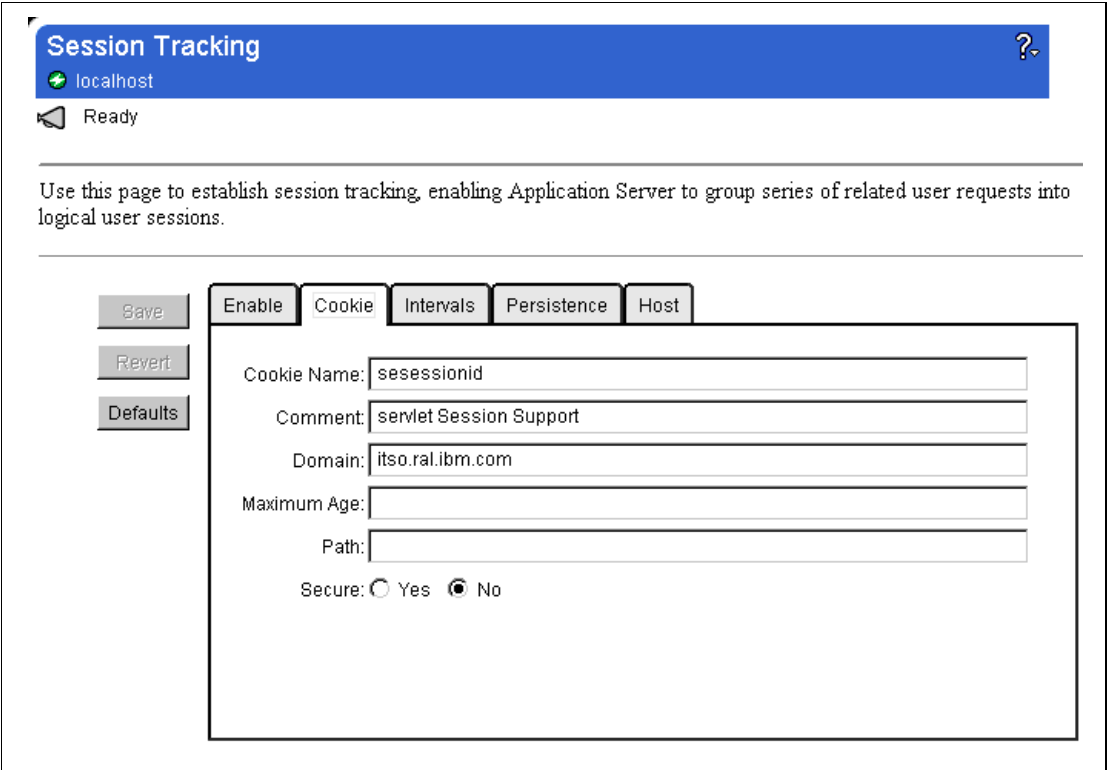


Figure 203. Cookie Tab from Setup > Session Tracking

The fields on the cookie page are described below:

Table 26. Fields on Cookie Tab

Field	Comment
Cookie Name	If cookies are enabled, specifies the name of the cookie. Default: session ID.
Comment	Comments about the cookie. Default: servlet Session Support.
Domain	If specified, defines the value of the domain field that is sent for session cookies. Specifies a value to restrict where session cookies will be sent. For example, if you specify a particular domain, session cookies will only be sent to hosts in that domain. Note: If you specify this field, the session object won't work properly from the access using "localhost" or IP address.
Maximum Age	If specified, defines the value of the maximum age of the cookie. Specifies a value to restrict or extend how long the session cookie will live on the client browser. The value is an integer that specifies the cookie age in milliseconds. Default: The cookie persists only for the current invocation of the browser. When the browser shuts down, the cookie is deleted.
Path	If specified, defines the value of the path field that will be sent for session cookies. Specifies a value to restrict which paths on the server (and therefore, which servlets, JHTML files, and HTML files) the cookies will be sent. By default, the path is "/" or the root directory, which means that the cookie will be sent on any access to the given server.

Field	Comment
Secure	Specifies whether session cookies include the secure field. Specifies a value to restrict the exchange of cookies to only HTTPS sessions. Default: No

5.1.2.4 Swapping

When many users access the Session Manage Application, the number of session objects is increased. That causes a reduction in the amount of memory resources available to your server.

WebSphere provides you with an interface to limit the number of session objects in memory. When the number of session objects generated exceeds the setting, WebSphere swaps out the least recently used session object to the hard disk.

For this setting, click the **Intervals** tab from the page **Setup -> Session Tracking**.

Figure 204. Intervals

Specify the fields:

Table 27. Fields on Interval Tab

Field	Default Value	Comment
Invalidation Interval	10000 (millisecond)	The amount of time between the checks that session tracking makes to determine if a session is no longer valid because it has not been used. The value must be an integer.

Field	Default Value	Comment
Swap Interval	10000 (millisecond)	The time between checks to see if we are past the maximum resident count and should start swapping the session out to disk.
Maximum Residents	1024	The number of sessions allowed to remain in memory at one time. If the number of sessions exceeds this number, sessions are swapped to disk on a least recently used basis to reduce the number of resident sessions.
Invalidate Time	1800000 (millisecond)	The amount of time a session is allowed to go unused before it is not validated. The value must be an integer.

Click **Save**.

For serialization technology in Java to be used in swapping, the data stored in the session object must implement `java.io.Serializable` or `java.io.Externalizable`.

5.1.2.5 Persistence

WebSphere provides you with a function called *persistence* to protect the session objects when the system is down.

The session values are swapped to disk when the Session Tracker shuts down and they are restored from disk when it restarts.

For persistence settings, open the session tracking page by clicking **Setup -> Session Tracking** from the Application Server Manager.

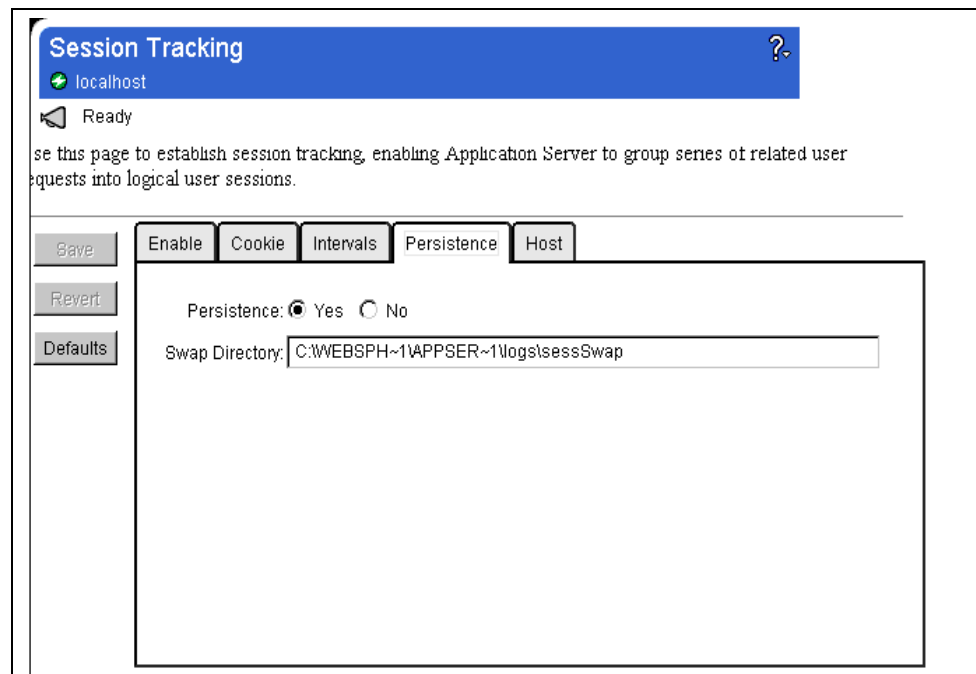


Figure 205. Session Tracking Persistence Setting

Table 28. The Field Summary on Persistence Tab

Field	Comment
Persistence	Whether session tracking keeps session data persistent. If Yes, sessions are swapped to disk when Session Tracker shuts down and are restored from disk when it restarts. If No, Session Tracker removes session swap files every time it starts. Default: Yes
Swap Directory	Name of the directory used to swap out session data. No other data should be kept in this directory. The default directory is <ASRoot>\logs\sessSwap.

If you specify a different directory from the Swap Directory field, the administration service will verify before saving that you have entered a fully qualified path name.

If it is not an absolute path name (for example, SwapTestDirectory for C:\SwapTestDirectory), the following message will appear:



Figure 206. Absolute Path Name Is Required

That means that the file does not already exist (it can be an existing directory name). See Figure 207:

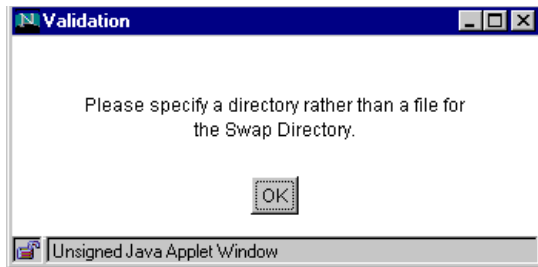


Figure 207. The Message When a File Name Is Specified Instead of the Directory

The directory can be written using the permissions granted to the IBM WebSphere Application Server, as shown in Figure 208 on page 233:

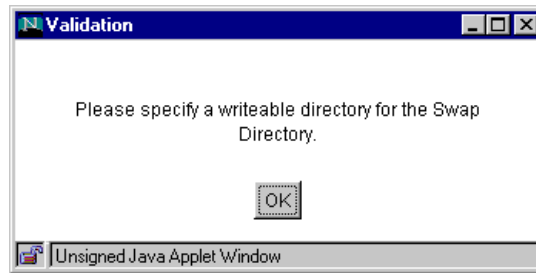


Figure 208. If the Permissions Are Not Granted

The default location for the swap directory is <ASRoot>\logs\sessSwap.

The following window shows how session objects are swapped out to a specific directory. When the Application Server goes down, the session objects are saved to these files.

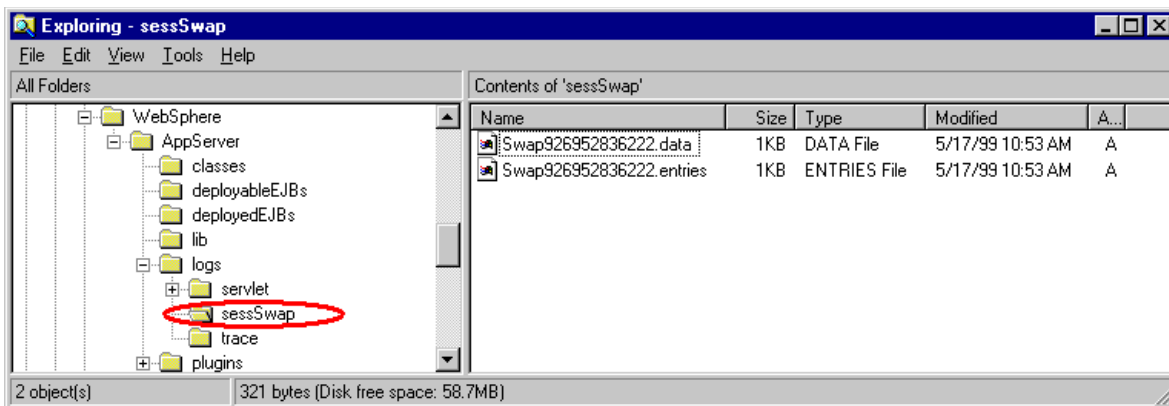


Figure 209. Swapped Out Session Objects

5.1.3 Session Object Sample

Here is a very simple example of session objects.

The first request to the SessionSample servlet generates a session object and stores an integer value in it. After the second request the servlet restores the integer value from the session object and increments it.

You can invoke the SessionSample servlet from Referrer.html. The SessionSample servlet, requested from the Web client, generates a session object and stores an integer value of 1 into an integer object called *intvalue*. After the second request from the same machine, the servlet looks up the session object (referring to its ID) and picks the *intvalue* out from the object. After incrementing it, it writes it back to the session object.

5.1.3.1 How to Work SessionSample Servlet

This part shows the steps required to work with this sample.

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html">
  <title>Referrer</title>
</head>
<body>
<H1>Referrer Page</H1>
<a href="/servlet/SessionSample">SessionSample</a><P>

<a
href="/servlet/IBMSessionSample">IBMSessionSample</a><
BR>

</body>
</html>
```

Figure 210. Referrer.html

1. Create the SessionSample.java and Referrer.html.

You can copy SessionSample.java from

<http://<hostname>//IBMWebAS/doc/howto/SessionSample.java.html>

For this HTML file, copy it from your browser window as shown in Figure 211:

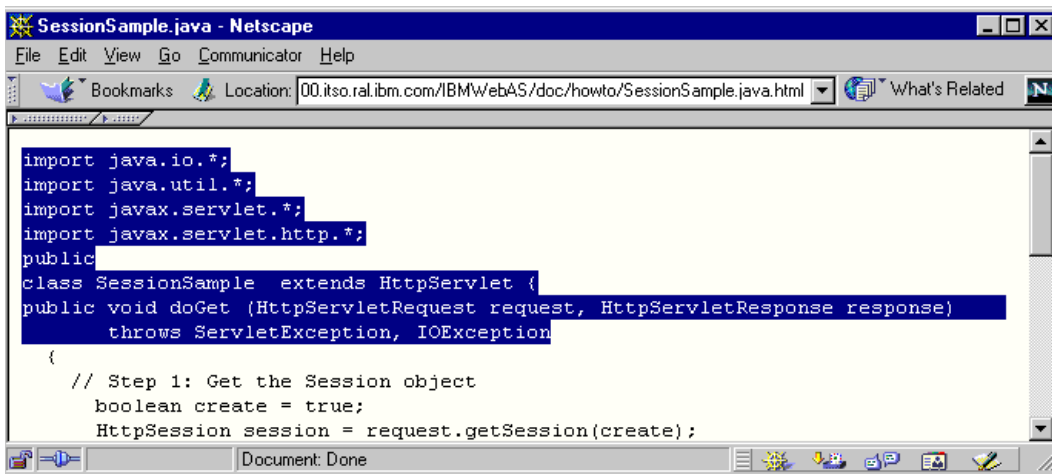


Figure 211. Copy and Paste from Your Browser

```

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.getValue ("sessiontest.counter");
    if (ival == null) ival = new Integer(1);
    else ival = new Integer (ival.intValue () + 1);
    session.putValue ("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>");
    }
}

```

Figure 212. SessionSample.java

2. Place the Referrer.html in your HTTP document root. For example, the default document root of IBM HTTP Server V1.3.3 on Windows NT is \Program Files\IBM HTTP Server\htdocs.
3. Place SessionSample.java in <ASRoot>/servlets. Compile SessionSample.java and create the class file.

```

C:\WEBSPH-1\APPSER-1\servlets>javac SessionSample.java
C:\WEBSPH-1\APPSER-1\servlets>

```

4. Open <http://<hostname>/Referrer.html>. The following window will appear:

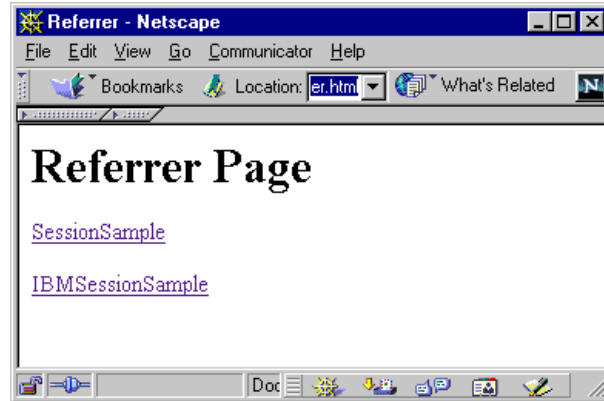


Figure 213. The Referrer Page

5. Click the **SessionSample** URL link. You get the following result sent to the requesting SessionSample servlet:

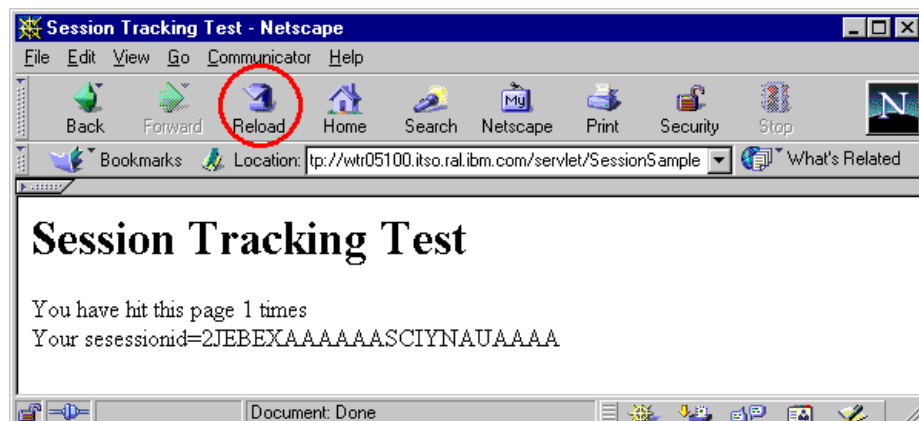


Figure 214. The Result of Session Sample

6. Click the **Reload** button on your browser. You get an incremented value each time you reload the window.

5.1.3.2 How to Use the IBMSessionSample Servlet

We show another example using the IBMSessionData, instead of HttpSession. It works similarly to SessionSample.java. Additionally, it uses set and getUsername() and set and getMessage() from the class com.ibm.servlet.personalization.sessiontracking.IBMSessionData.


```

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.ibm.servlet.personalization.sessiontracking.IBMSessionData;

public class IBMSessionSample extends HttpServlet {
public void doGet (HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException
    {
    // Step 1: Get the Session object casting to IBMSessionData
    boolean create = true;
    IBMSessionData session =(IBMSessionData)request.getSession(create);
    // Step 2: Get the session data value
    Integer ival = (Integer) session.getValue ("IBMsessiontest.counter");
    String name=session.getUserName(); //Specific to IBMSessionData
    String message=(String)session.getMessage(); //Specific to
IBMSessionData

    if (ival == null){
    ival = new Integer (1);
    //Specific to IBMSessionData
    session.setUserName("YourName");
    session.setMessage("You can put any object that implements
Serializable");
    }
    else{
    ival = new Integer (ival.intValue () + 1);
    }
    session.putValue ("IBMsessiontest.counter", ival);
    // Step 3: Output the page
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head><title>IBM Session Tracking Test</title></head>");
    out.println("<body>");
    out.println("<h1>IBM Session Tracking Test</h1>");
    out.println ("You have hit this page " + ival + " times" + "<br>");
    out.println ("Your " + request.getHeader("Cookie")+"<BR>");

    out.println ("User Name: " + name + "<BR>");
    out.println ("Message: " +message+"<BR>");
    out.println("</body></html>");
    }
}

```

Figure 215. IBMSessionSample.java

This sample used Referrer.html. Before performing the following steps, steps 1 and 2 from 5.1.3.1, “How to Work SessionSample Servlet” on page 233 are required:

1. Create IBMSessionSample.java.
2. Place IBMSessionSample.java in <ASRoot>/servlets. Compile IBMSessionSample.java and create the class file.

```
C:\WEBSPH~1\APPSER~1\servlets>javac IBMSessionSample.java
C:\WEBSPH~1\APPSER~1\servlets>
```

3. Open `http://<hostname>/Referrer.html`. The following window appears:



Figure 216. The Referrer Page

4. Click the **IBMSessionSample** URL link. The first time you try it you get the following result. The *User Name* is anonymous because it has not been set yet. The same is true for Message.

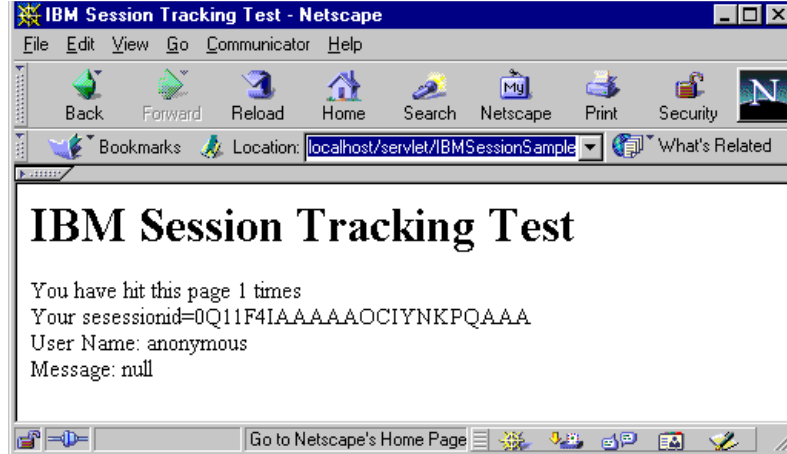


Figure 217. The Result of IBMSessionSample (1)

5. After the second request (click the **Reload** button on your browser), you can see that User Name and Message changed, as shown in the following window:

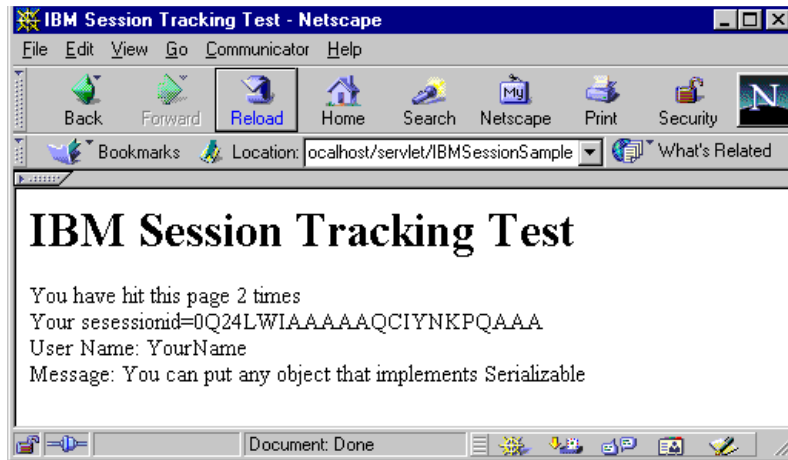


Figure 218. The Result of IBMSessionSample (2)

6. Click **Server Execution Analysis -> Monitors -> Active Sessions** from the Application Server Manager. You can see UserName set to the YourName property of the session object.

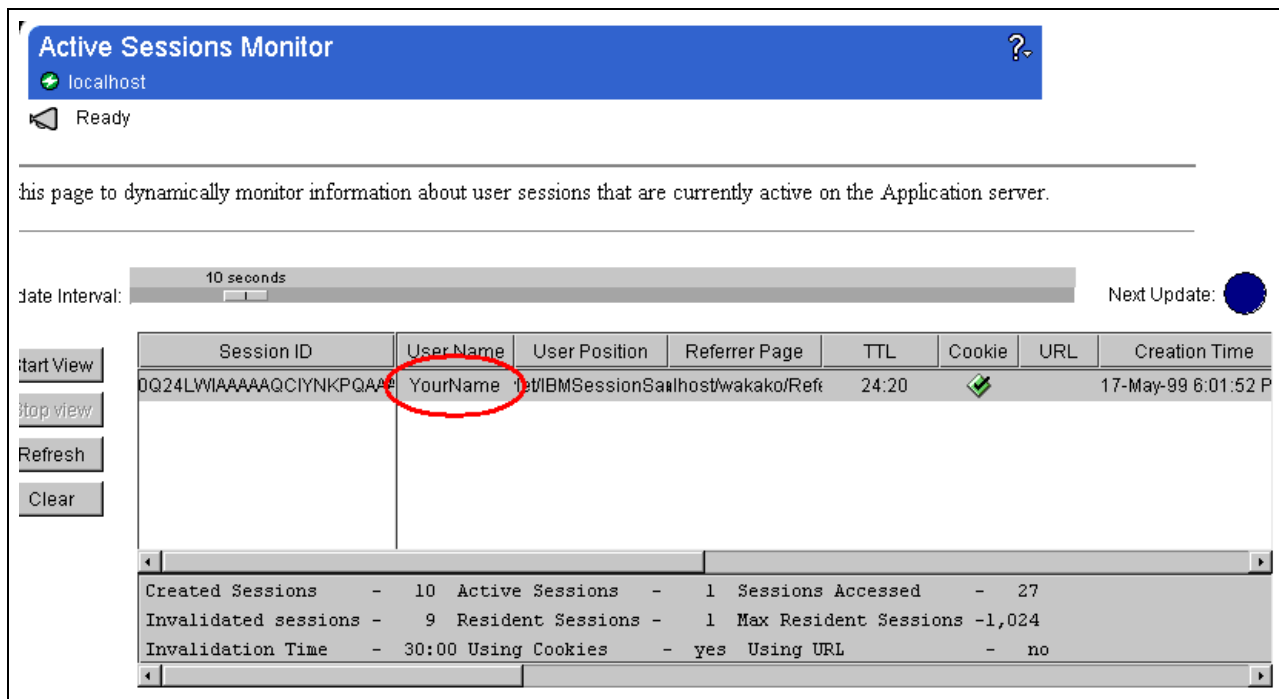


Figure 219. "YourName" Was Set to User Name Field

5.1.3.3 Session Object Using URL Rewriting

Cookies are not necessarily valid on all browsers. Some browsers can't use cookies and some browsers disable cookies. If you can't use cookies you can use URL rewriting to carry the session ID. To use URL rewriting, specific coding is necessary (in addition to configuring the Application Server Manager), as shown in Figure 220 on page 240. If the engine sends a cookie in the client's incoming request, the URL will not be encoded with the session ID.

Configuration for URL Rewriting

To enable URL rewriting:

1. Click **Setup** -> **Session Tracking** from the Application Server.
2. Open the **Enable** tab on the Session Tracking page.
3. Check the **Yes** option to enable URL rewriting, and click **Save**.

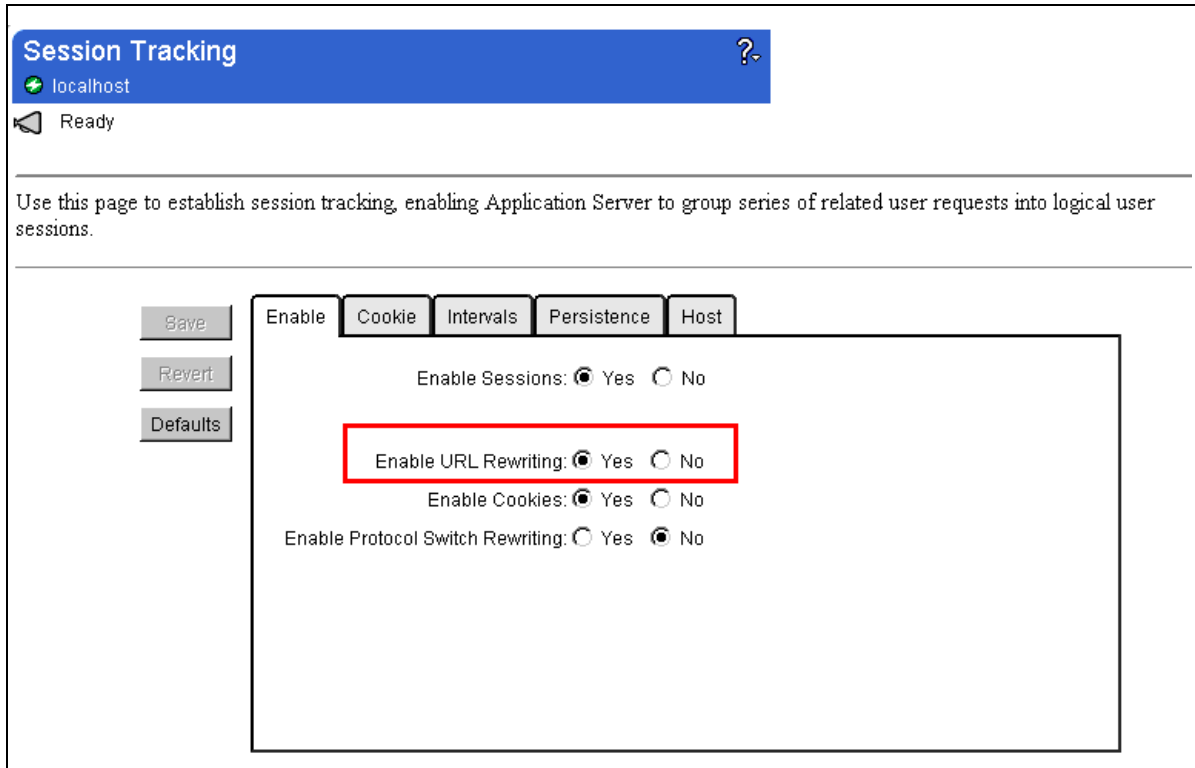


Figure 220. Enable URL Rewriting

URL Rewriting Sample

This sample uses three files:

1. URLRewrite.html calls the URLRewriteServlet1 servlet passing the string in the memo field.
2. URLRewriteServlet1.java processes the request, creates the session object and outputs HTML with a link to the URL RewriteServlet2 servlet with `res.encodeRedirectUrl(rewriteURL)`. This method is indispensable to managing sessions using URL rewriting.
3. URLRewriteServlet2.java gets the memo and session ID from the session object. This servlet doesn't require URL rewriting specific coding.

```
<html><head>
<title>Test of URL Rewrite Servlet </title></head>
<body>
<form method=GET action=/servlet/URLRewriteServlet1>
<H1>
Please input your message!
</h1>
<input type="text" name="memo">
<input type="submit" value="Submit">
</form>
</body>
</html>
<!-- hhmts start -->
Last modified: Fri Jun 04 14:12:59 1999
<!-- hhmts end -->
```

Figure 221. *URLRewrite.html*

```

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class URLRewriteServlet1 extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {

        HttpSession session = req.getSession(true);
        PrintWriter out = res.getWriter();
        res.setContentType("text/html");
        out.println("<html><head><title>This is a URLRewrite Servlet.
        </title></head>");
        out.println("<body>");
        out.println("<hr>SessionID : " + session.getId());

        String memo = (String) session.getValue("memo");
        if (memo == null){
            out.println("<H2>There is no memo.</H2>");
        }
        else{
            out.println("<hr> The last memo is<B> " + memo + "</B><BR>");
        }
        memo =req.getParameterValues("memo") [0];
        session.putValue("URLRewrite.memo",memo);
        session.putValue("OriginURL", req.getRequestURI());
        out.println("<h1> Now, the new memo is " + memo);
        out.println("</h1>");
        out.println("<a href=\"");
        String rewriteURL = "/servlet/URLRewriteServlet2";

        out.println(res.encodeRedirectUrl(rewriteURL));

        out.println("<h1> Go to URLRewrite Servlet2 </a>");
        out.println("<p>");
        out.println("<hr>");
        out.println("<a href=\"/URLRewriteServlet.html\"> Go back to URLrewrite
        HTML</a>");
        out.println("</body></html>");
        out.flush();
    }
}

```

Figure 222. URLRewriteServlet1.java

```

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class URLRewriteServlet2 extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        HttpSession session = req.getSession(false);
        String originURL = (String) session.getValue("OriginURL");
        String memo = (String) session.getValue("URLRewrite.memo");

        out.println("<html><head><title>URLRewriteServlet2 </title></head>");
        out.println("<body>");
        out.println("<h1> URLRewriteServlet2 </h1>");
        out.println("<P>");
        if (memo == null){
            out.println("<hr>There is no memo.");
        }
        else{
            out.println("<hr> The last memo is <B>" + memo+"</B>");
        }
        out.println("<p>");
        out.println("<hr> Session ID : " + session.getId());
        out.println("<P>");
        out.println("<hr>");
        out.println("<a href=\"/URLRewrite.html\"> Go back to URLrewrite
HTML</a>");
        out.println("</body></html>");
    }
}

```

Figure 223. URLRewriteServlet2.java

To use this sample:

1. Create URLRewrite.html, URLRewriteServlet1.java and URLRewriteServlet2.java.
2. Place URLRewrite.html in the servlets directory.
3. Compile URLRewriteServlet1.java and URLRewriteServlet2.java. Place the class files to <ASRoot>/servlets.

```

C:\WebSphere\AppServer\Servlets>javac URLRewriteServlet1.java
C:\WebSphere\AppServer\Servlets>javac URLRewriteServlet2.java
C:\WebSphere\AppServer\Servlets>

```

4. Open <http://<hostname>/URLRewrite.html>.

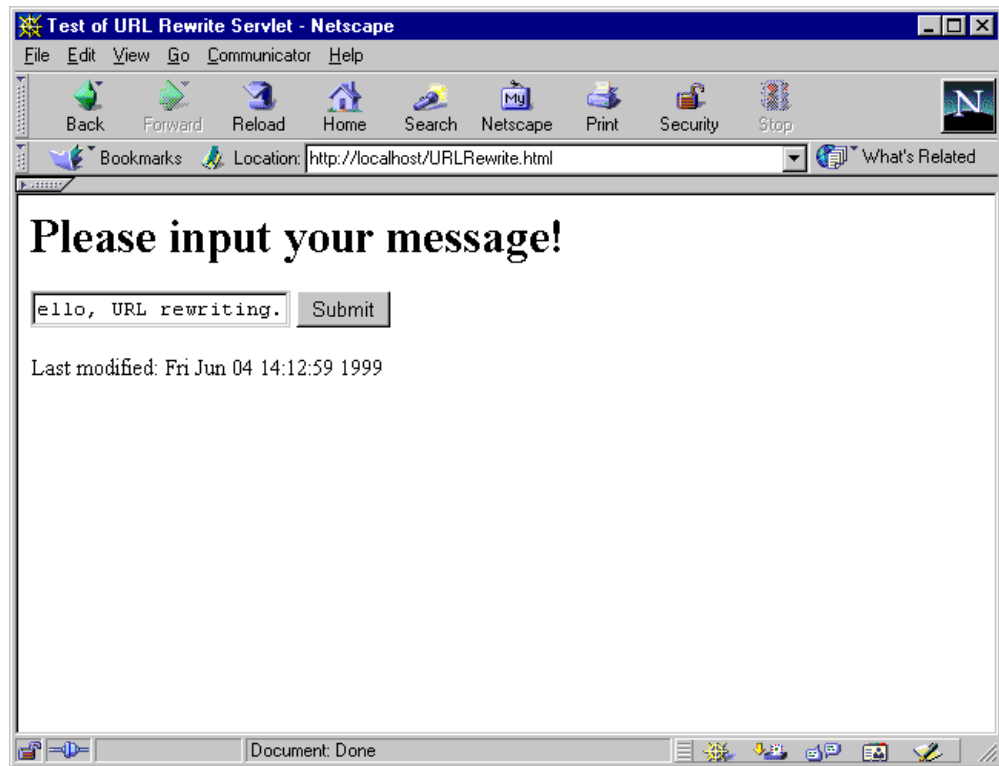


Figure 224. Referred URLRewrite.html

5. Input the message and click the **Submit** button. The following window will appear. You can see the ID of the created session in this window (Figure 225 on page 245). Click the **Go to URLRewriteServlet2** link.



Figure 225. The Result of URLRewriteServlet1

6. You can see the message was stored from session object (Figure 226 on page 246). This way the session is managed with the requests. The session ID is added to the URL because the URLRewriteServlet1 uses URL rewriting.

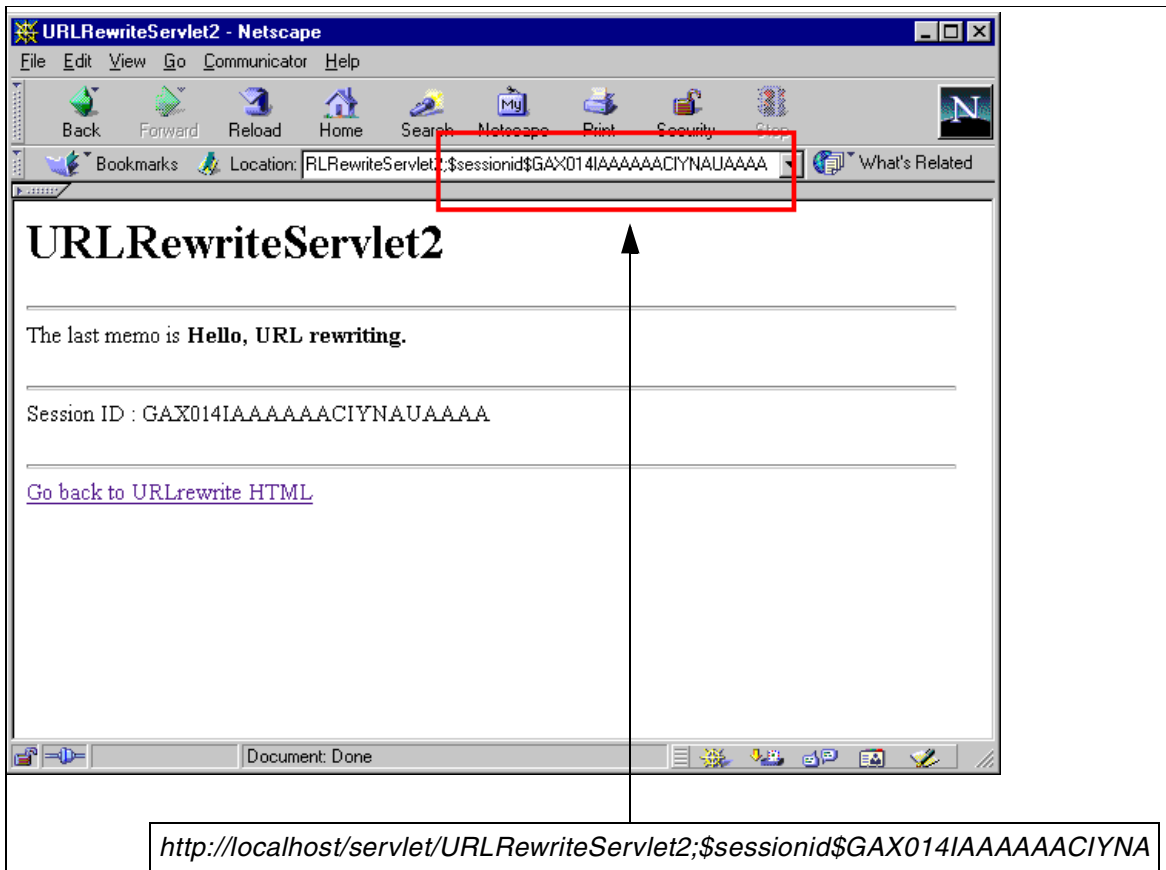


Figure 226. The Result of URLRewriteServlet2

7. Click **Server Execution Analysis -> Monitors -> Active Sessions** from the Application Server Manager. You can make sure that URL rewriting was used if you see the green check box under the URL field in the following window:

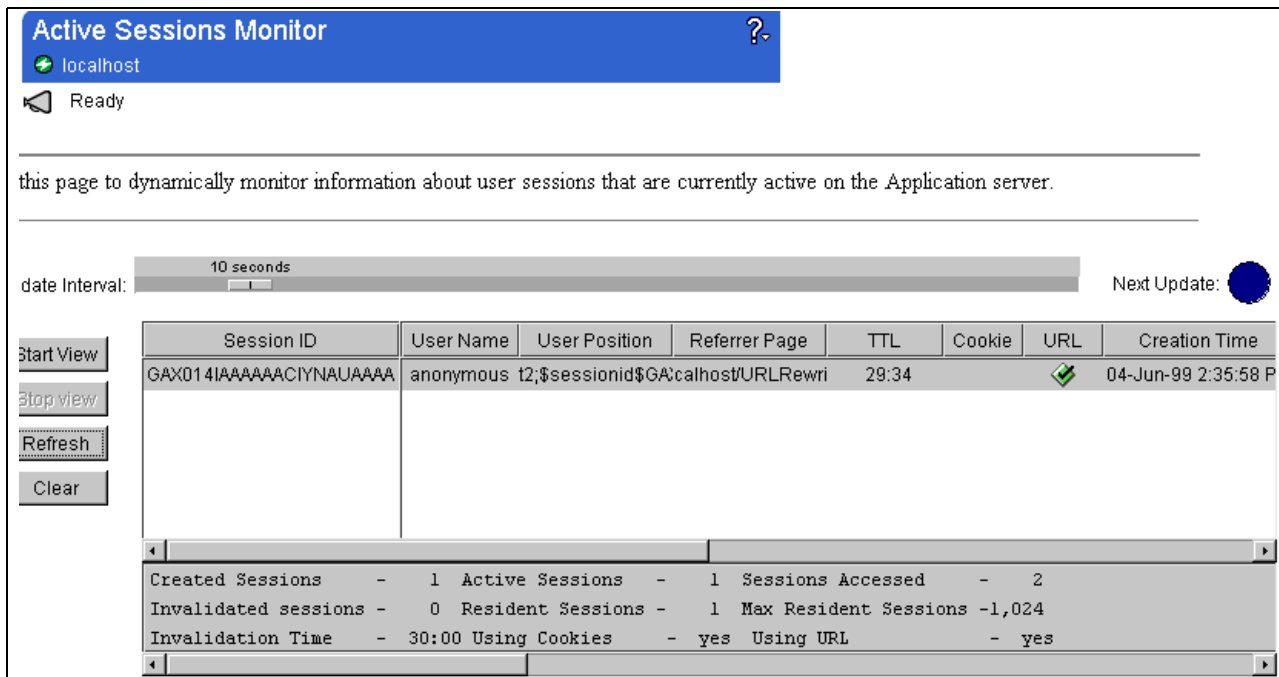


Figure 227. URL Was Used

- If your browser can accept cookies (Figure 228 on page 247), you can see this application using cookies instead of URL rewriting (Figure 229 on page 248):

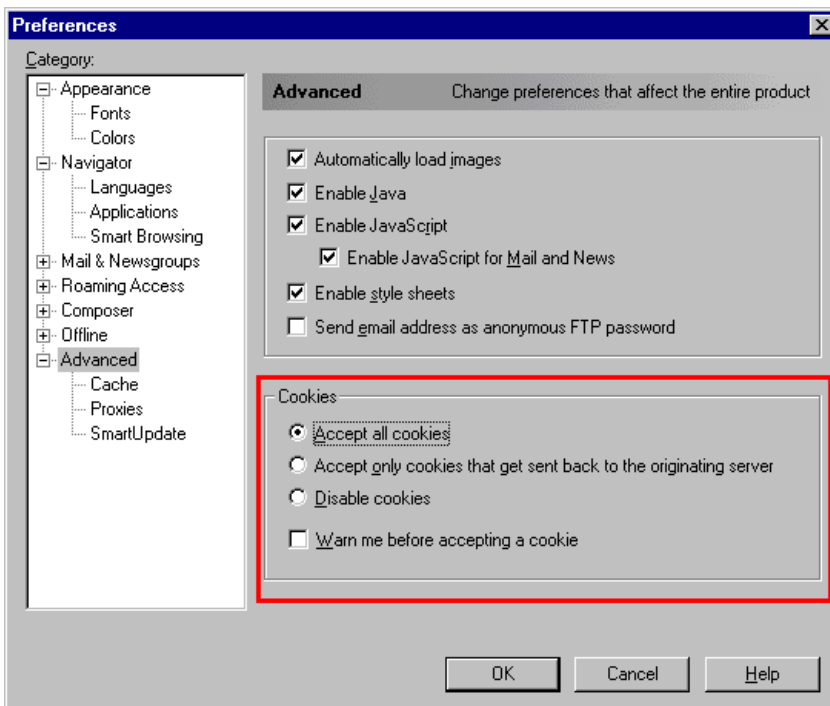


Figure 228. Cookie Option Is Enabled in Netscape

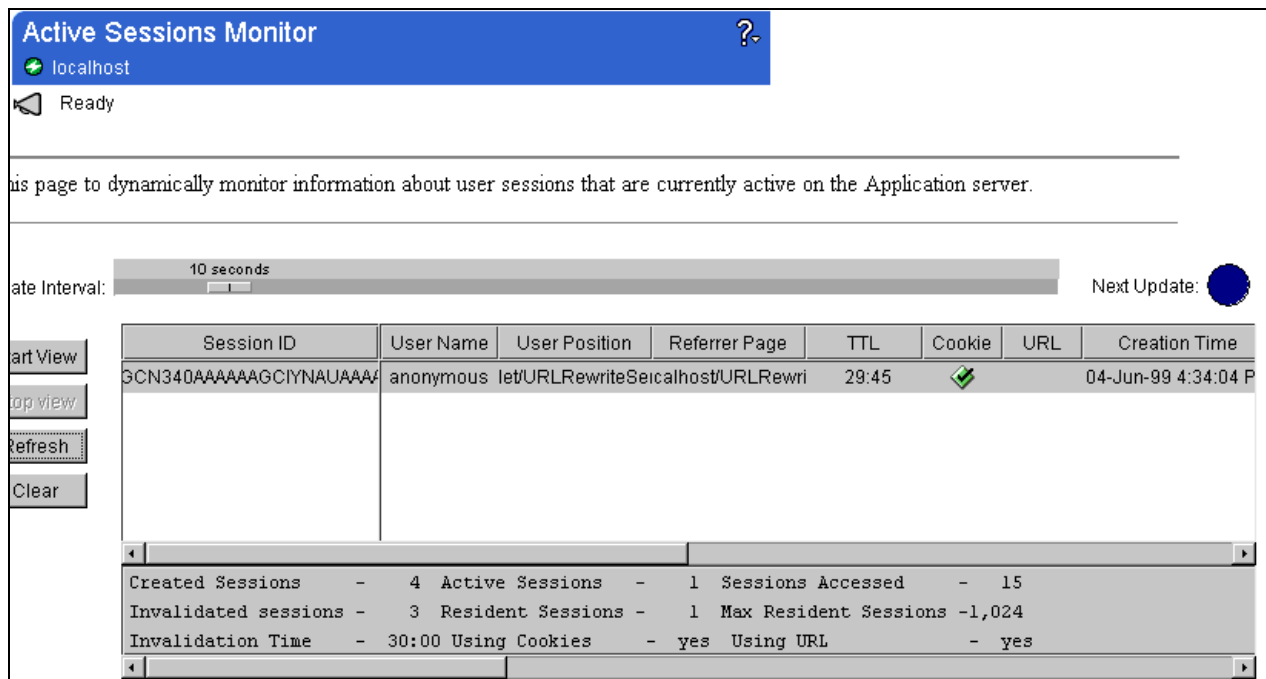


Figure 229. Cookie Is Used

5.1.4 Session Clustering

Session clustering is the technology that is used to share session objects among multiple application servers. This is accomplished by one application server, called a cluster *server*, maintaining information about the session objects, and another application server, called a cluster *client*, asking for information from the cluster server. The class that handles these processes is *com.ibm.servlet.personalization.sessiontracking.IBMSessionContextImpl*. It is also called the Session Tracker.

Specific coding for clustering is not necessary.

Class files get added to the session object for the WebSphere Application Server class path. In the session clustering environment, the class file for session objects should be referred to in every machine in the cluster. To build a clustering environment, eNetwork Dispatcher in WebSphere Performance Pack is required. WebSphere Performance Pack enhances and manages the performance of WebSphere Web sites. See 1.1.3, “WebSphere Performance Pack” on page 5 for more details.

5.1.4.1 Session Management Modes

There are three models in which session support operates:

1. Stand-Alone

This is the default setting. The server maintains its own session information. It does not request session information about the other servers.

2. Session Cluster Client

One of the WebSphere Application Servers that uses session clustering.

3. Session Cluster Server

The WebSphere Application Server host that maintains the information for all session objects.

To see the settings for each server click **Setup -> Session Tracking** from the Application Server Manager. Click the **Host** tab. The following setting will be on the Host Page of the Session Tracking.

Stand-Alone

This is the default setting. The Stand Alone Host option is checked **Yes** and the other fields are left blank (see Figure 230).

Click the **Save** button and restart the application server for the changes to take place.

Session Tracking
localhost
Ready

Use this page to establish session tracking, enabling Application Server to group series of related user requests into logical user sessions.

Save Revert Defaults

Enable Cookie Intervals Persistence Host

Stand Alone Host (change requires restart): Yes No

Session Cluster Server: (change requires restart)

Protocol Switch Host:

Figure 230. Setting for Stand Alone

Session Cluster Server

For the session cluster server setting:

1. Check the **No** option on the Stand Alone Host option (Figure 231). Click the **Save** button and restart the application server.

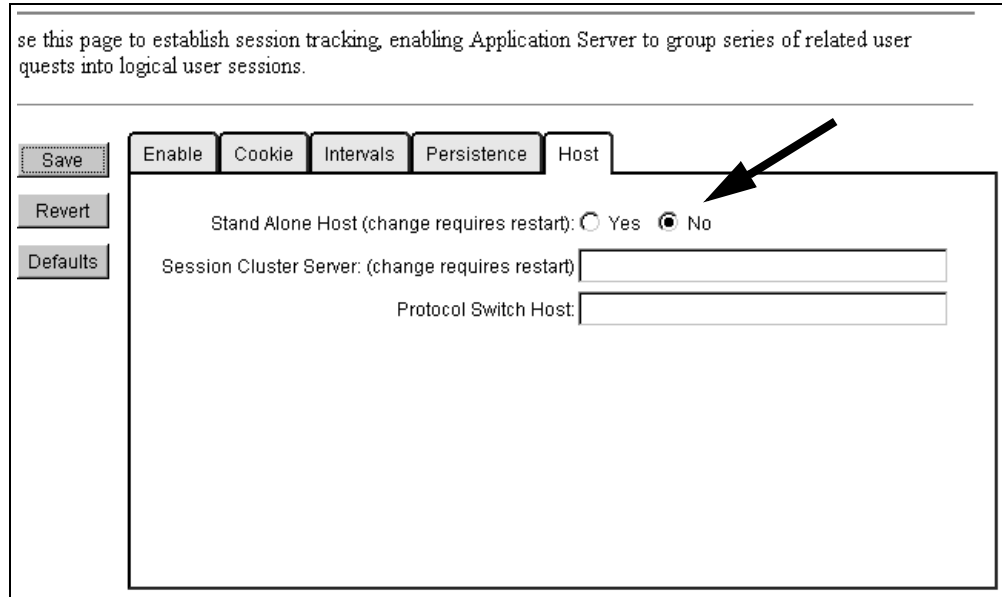


Figure 231. Check the Option Stand Alone - No

2. Leave the Session Cluster Server field blank. That sets up this instance of the IBM WebSphere Application Server as the session cluster server.

Session Cluster Client

To set up the Session cluster client:

1. Check the **No** option on the Stand Alone Host option (Figure 231 on page 250). Click the **Save** button and restart the application server.

If you try to make further changes before restarting the application server, the following dialog box will come up, and the setting will be lost after restarting:

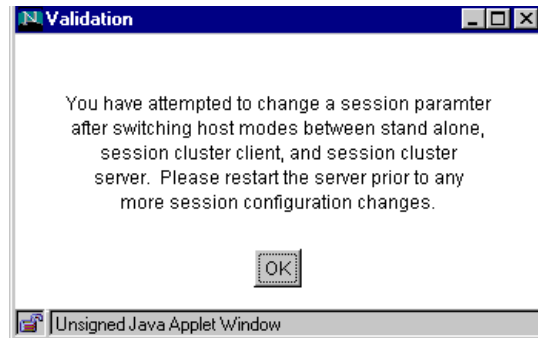


Figure 232. Dialog Box for Validation

2. Restart the WebSphere Application Server.
3. Open the Host tab on the Session Tracking page (click **Setup -> Session Tracking** from the Application Server Manager and click the **Host** tab).
4. In the Session Cluster Server field, type the host name or IP address of another instance of the IBM WebSphere Application Server that is set up as a cluster server (Figure 233).

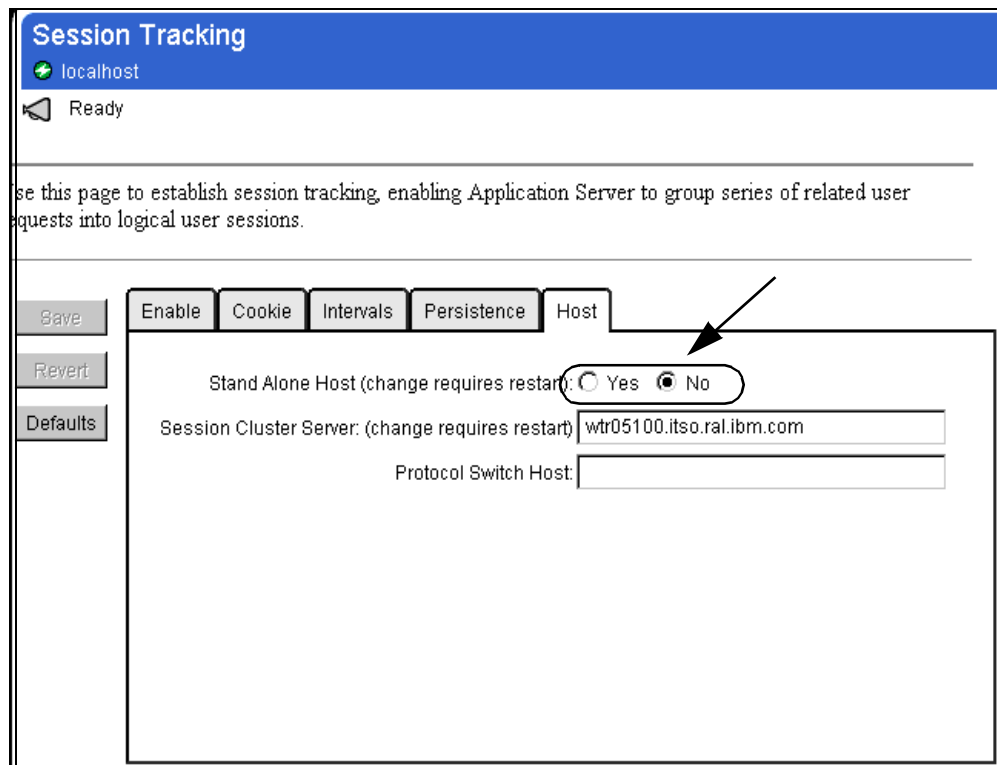


Figure 233. The Cluster Client Setting

5. Restart the WebSphere Application Server.

5.1.4.2 Session Clustering Scenario

Figure 234 on page 252 shows the flow for session clustering. This example shows the flow when the requested session object exists in a different server from the one that processed the request from the client.

1. The session object is generated when *HttpServletRequest.getSession(true)* is called. The generated session object is stored on the server where it was generated, whether it was a clustered server or not.
2. WAS (WebSphere Application Server) registers the generated session object at the cluster server, sending information about the session object.
3. The cluster client that requested the session object from the Web client doesn't know where it is. Therefore, the cluster client asks for the location from the cluster server.
4. The cluster server notifies the cluster client of the WAS host that is maintaining the requested session object.
5. The WAS host that processed the request from the Web client requests the session object from the WAS host that maintains the requested session object.
6. The WAS host maintaining the requested session object locks the session object.
7. The WAS host maintaining the requested session object sends the session object to WAS.

8. When the service() method that processes the Web client request is terminated, notification of the termination is sent to the WAS host maintaining the session object with refreshed object information.
9. The WAS host unlocks the requested session object and applies changes to the session object.

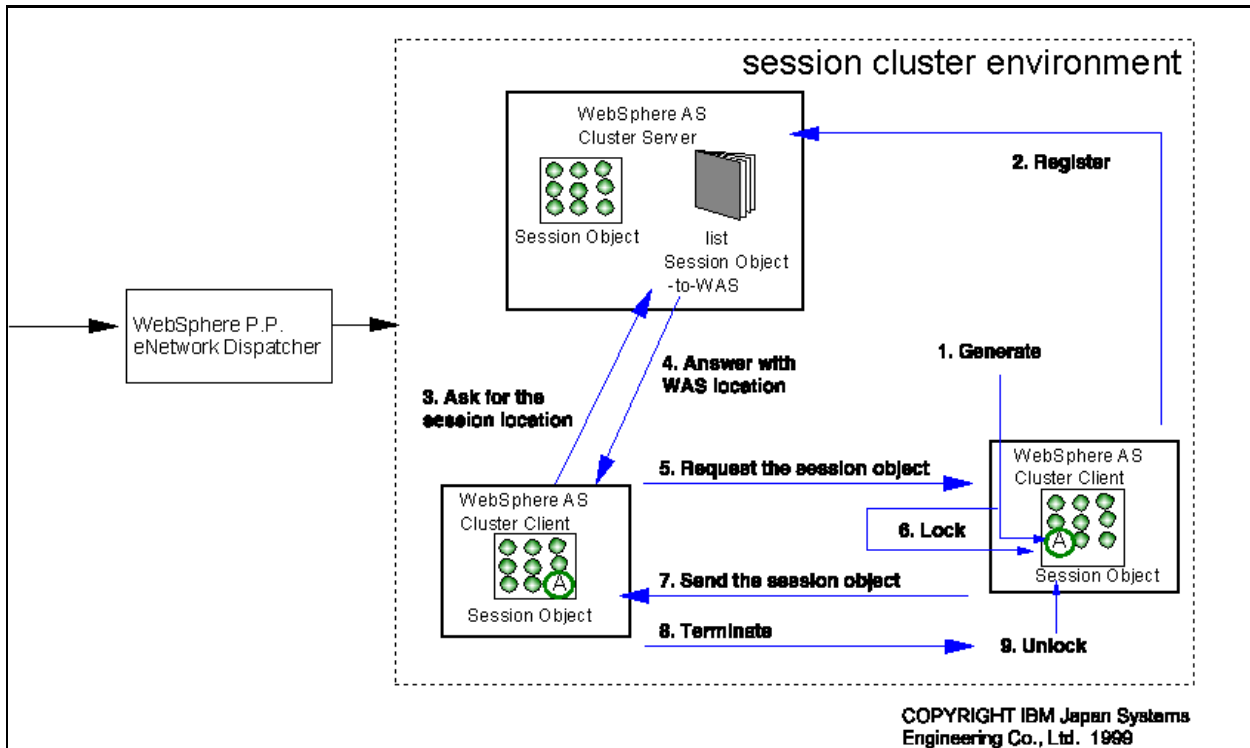


Figure 234. Session Clustering Flow

If the requested session object is stored at the host that processes the request from the Web client, it doesn't ask the location of the session object from the cluster server.

Note: The parameters for the cluster server are used for the clustered environment. To be consistent with the clustered environment, the valid parameters in the cluster's client are limited to the *Enable Sessions* parameter in the Enable tab (**Setup -> Session Tracking** from the Application Server Manager as shown in Figure 202 on page 228) and the Persistence tab (Figure 205 on page 231).

The object placed in the session data must implement the serializable interface (Figure 235 on page 253) to propagate the object along with a given session since the session is serialized across the cluster.


```

public class ShoppingCart implements java.io.Serializable{
:
:
}

```

Figure 235. Objects Placed in Session Data Must Implement the Serializable Interface

5.2 User Profiles

The Application Server has an API called UserProfile that makes it easy to maintain persistent information about your Web site visitors. The UserProfile enables you to store the user's data in a database without coding any SQL. The UserProfile contains the interface to store the user's personal data, such as name, postal and e-mail addresses, telephone numbers and shopping cart information. For further information, see the UserProfile API Reference:

<http://<hostname>/IBMWebAS/doc/apidocs/Package-com.ibm.servlet.personalization.userprofile.html>

5.2.1 Setting Up User Profiles

Before setting up the user profile, you must set up a JDBC connection to the database. Refer to 2.3.5.1, "Setting Up DB2 in the WebSphere Environment" on page 74.

1. Click **Setup -> User Profile** from the Application Server Manager.
2. Click the **Enable** tab and select **Yes** for the Using User Profiles? option.
3. Leave the Class Name field with its default value. If you want to use a created subclass for UserProfile, specify the absolute class name.

The screenshot shows the 'User Profile' configuration page. At the top, there is a blue header with the text 'User Profile' and 'localhost'. Below the header, there is a 'Ready' status indicator. The main content area contains a description: 'Use this page to configure the User Profile class, which defines and maintains information about the people who visit the Web site.' Below the description, there are three tabs: 'Enable', 'Database', and 'ConnMgr'. The 'Enable' tab is selected. Under the 'Enable' tab, there are three buttons: 'Save', 'Revert', and 'Defaults'. The 'Using User Profiles?' section has two radio buttons: 'Yes' (which is selected and circled in red) and 'No'. Below this, the 'Class Name' field contains the text 'com.ibm.servlet.personalization.userprofile.UserProfile'.

Figure 236. User Profile Page

4. Click the **Database** tab.

User Profile
localhost

Ready

Use this page to configure the User Profile class, which defines and maintains information about the people who visit the Web site.

Save Enable Database ConnMgr

Revert

Defaults

Database Used: db2

JDBC Driver used: COM.ibm.db2.jdbc.app.DB2Driver

Database Name: wbsphere

Database Owner: db2inst1

Table Name: userprofile

User ID: db2inst1

Password: *****

Figure 237. Database Tab on User Profile Page

5. Fill in the fields. You must specify an existing database name in the Database Name field. You shouldn't create a userprofile table in the database. It will be created automatically for you when the first request to the user profile occurs.

Table 29. Field Summary on Database Tab on User Profile Page

Field Name	Default Value	Comment
Database Used	db2	The database product name (for example, DB2 or Oracle). This information is used in creating the JDBC connection to the database.
JDBC Driver Used	COM.ibm.db2.jdbc.app.DB2Driver	The name of the JDBC driver for the database. Note: Include the.zip or .jar file for the driver (such as db2java.zip for DB2) in the Java classpath of the Application Server (see the Paths tab on the Java Engine page to set the classpath).
Database Name	N/A	The database used to store the user profile tables and data. You must create this database before you use user profile first, or specify an existing database. But you need not create any tables in the database.
Database Owner	N/A	The ID of the owner of the database.

Field Name	Default Value	Comment
Table Name	userprofile	The table to store user profile data. The user profile class will create this table using each of these applicable configuration variables when it is first enabled. You need only specify a name for the table.
User ID	N/A	The user ID used to access the database and its tables.
Password	N/A	The password associated with the defined user ID.

6. Click the **ConnMgr** tab.

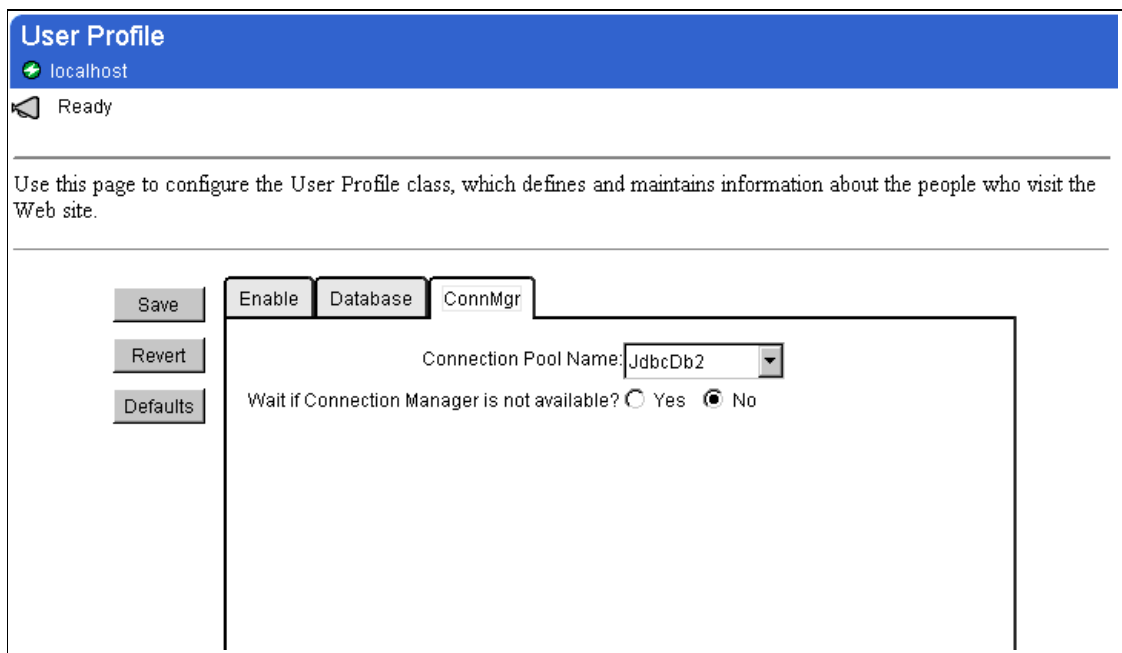


Figure 238. ConnMgr Tab on User Profile Page

7. Click the **Save** button.

5.2.2 How to Use UserProfile in Your Servlet

The following steps show the easiest process to follow to use the user profile in a servlet:

1. Import userprofile packages.

```
import com.ibm.servlet.personalization.userprofile.*;
```

2. Create the UserProfile object.

```
UserProfile up = new UserProfile();
```

3. Use the UserProfile.retrieveUserProfileByUserName() method to get the UserProfile object referenced by the specified userName.

```
(up = UserProfile.retrieveUserProfileByUserName(userName)
```

Many methods are provided to get the UserProfile object, for example, UserProfile.retrieveUserProfileByXXXX. See the UserProfile API reference for more details:

<http://<hostname>//IBMWebAS/doc/apidocs/com.ibm.servlet.personalization.userprofile.UserProfile.htm>

4. Use the UserProfile.addUserProfile() method to create a new user profile for a user:

```
up = UserProfile.addUserProfile(userName);
```

5. Use the UserProfile.updateUserProfile() method to update a user profile for a user:

```
Hashtable userInfo = new Hashtable();
userInfo.put("userName", userName);
userInfo.put("email", req.getParameterValues("email")[0]);
userInfo.put("dayPhone", req.getParameterValues("dayPhone")[0]);
userInfo.put("fax", req.getParameterValues("fax")[0]);
userInfo.put("address1", req.getParameterValues("address1")[0]);
up.updateUserProfile(userInfo);
```

6. Use UserProfile.getXXXXX() method to get the data of the user profile:

```
pw.println("<tr><td>userName:</td><td>"+up.getUserName()+"</td></tr>");
pw.println("<tr><td>email :</td><td>"+up.getEmail()+"</td></tr>");
pw.println("<tr><td>dayPhone :</td><td>"+up.getDayPhone()+"</td></tr>");
pw.println("<tr><td>fax :</td><td>"+up.getFax()+"</td></tr>");
pw.println("<tr><td>address1 :</td><td>"+up.getAddress1()+"</td></tr>");
```

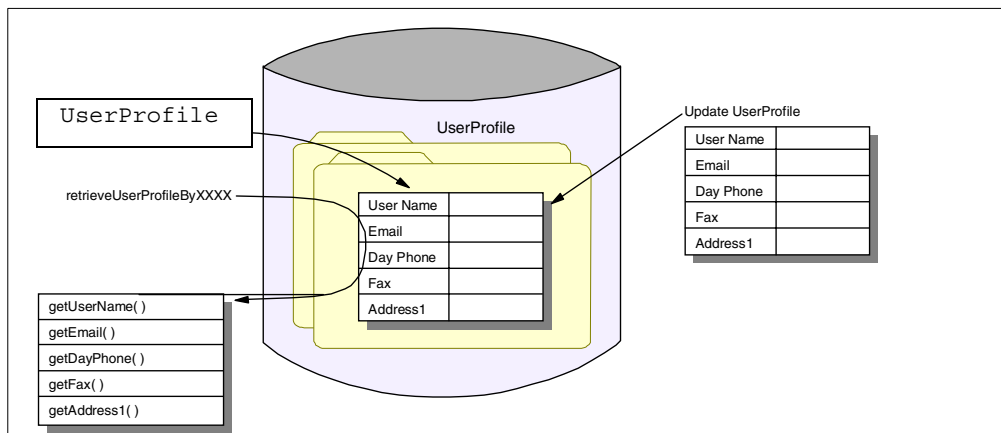


Figure 239. retrieveUserProfileByXXXX

5.2.3 UserProfile Sample

We show a simple example on how to use UserProfile. A user can retrieve or add data for a specified user in HTML. A servlet processes the request from the HTML and accesses the UserProfile.

To work with the User Profile sample:

1. Create UserProfileSample1.html and UserProfileSample1.java:

```
<html> <head>
<title>User Profile Sample1</title>
</head>

<BODY TEXT="#000000" BGCOLOR="#eeffee">
<h1>User Profile Sample1</h1>
<p><br>

<form action = "http://localhost/servlet/UserProfileSample1" method =
"POST">
<table COLS=2 >
<tr><td>User Name:</td>
<td><input type = "text" name = "userName"></td></tr>

<tr><td>email:</td>
<td><input type = "text" name = "email"></td></tr>

<tr><td>dayPhone:</td>
<td><input type = "text" name = "dayPhone"></td></tr>

<tr><td>fax:</td>
<td><input type = "text" name = "fax"></td></tr>

<tr><td>address1:</td>
<td><input type = "text" name = "address1"></td></tr>

<tr><td><INPUT TYPE="submit" NAME="button" VALUE="retrieve" ></td>
<td><INPUT TYPE="submit" NAME="button" VALUE=" add " ></td></tr>
</table>
</form>
<hr>
<!-- hhmts start -->
Last modified: Mon May 24 13:03:50 1999
<!-- hhmts end -->
</body> </html>
```

Figure 240. UserProfileSample1.html

```

import com.ibm.servlet.personalization.sessiontracking.*;
import com.ibm.servlet.personalization.userprofile.*;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class UserProfileSample1 extends HttpServlet {
    public void doPost(javax.servlet.http.HttpServletRequest
req, javax.servlet.http.HttpServletResponse res) throws ServletException,
IOException {
        String button = null;
        String userName = null;
        String message = null;

        UserProfile up = new UserProfile();
        Hashtable userInfo = new Hashtable();

        button = req.getParameterValues("button")[0];
        userName = req.getParameterValues("userName")[0];

        if (button.equals("retrieve")){

            if((up = UserProfile.retrieveUserProfileByUserName(userName)) ==
null){
message = "No Data Exists for <B> "+userName+" </B>";
            }else{
message = "Search Finished";
            }

            }else{
                if((up = UserProfile.retrieveUserProfileByUserName(userName)) ==
null){
up = UserProfile.addUserProfile(userName);
message = "New User was Resistered";
                }else{
message = "Updated User Info";
                }
            }
        }
    }
}

```

Figure 241. *UserProfileSample1.java (1/2)*

```

userInfo.put ("userName", userName);
    userInfo.put ("email", req.getParameterValues ("email") [0]);
    userInfo.put ("dayPhone", req.getParameterValues ("dayPhone") [0]);
    userInfo.put ("fax", req.getParameterValues ("fax") [0]);
    userInfo.put ("address1", req.getParameterValues ("address1") [0]);

    up.updateUserProfile (userInfo);
}
res.setContentType ("text/html");
res.setHeader ("Pragma", "No-cache");
res.setHeader ("Cache-Control", "no-cache");
res.setDateHeader ("Expires", 0);

PrintWriter pw = res.getWriter ();

pw.println("<HTML><HEAD>");
pw.println("<TITLE>The Result of UserProfileSample1</TITLE></HEAD>");
pw.println("<BODY TEXT=\\"#000000\\" BGCOLOR=\\"#eeffee\\">");
pw.println("<H1>The Result of UserProfileSample1</H1>");
pw.println("<HR WIDTH=\\"100%\\">");
pw.println("<BR>" + message);
pw.println("<FORM ACTION = \\"http://localhost/userProfileSample1.html\\"
METHOD = \\"GET\\">");
    if (up != null) {
        pw.println("<table COLS=2 ><B>");

pw.println("<tr><td>userName:</td><td>" + up.getUserName () + "</td></tr>");
        pw.println("<tr><td>email :</td><td>" + up.getEmail () + "</td></tr>");
        pw.println("<tr><td>dayPhone
:</td><td>" + up.getDayPhone () + "</td></tr>");
        pw.println("<tr><td>fax :</td><td>" + up.getFax () + "</td></tr>");
        pw.println("<tr><td>address1 :</td><td>"
+ up.getAddress1 () + "</td></tr>");
        pw.println("</B></table>");
    }
    pw.println("<P><INPUT TYPE = \\"submit\\" VALUE = \\"Back\\"></FORM>");
pw.println("</BODY></HTML>");

    pw.flush ();
    pw.close ();
}
}

```

Figure 242. UserProfileSample1.java (2/2)

2. Place UserProfileSample1.html in your HTTP document root. For example, the default document root for IBM HTTP Server V1.3.3 on Windows NT is \Program Files\ IBM HTTP Server\htdocs.
3. Place UserProfileSample1.java in <ASRoot>/servlets. Compile UserProfileSample1.java and create the UserProfileSample1.class file.

```
C:\WEBSPH~1\APPSER~1\servlets>javac UserProfileSample1.java
C:\WEBSPH~1\APPSER~1\servlets>
```

4. Open the URL `http://<hostname>/UserProfileSample1.html`. The following window will appear:

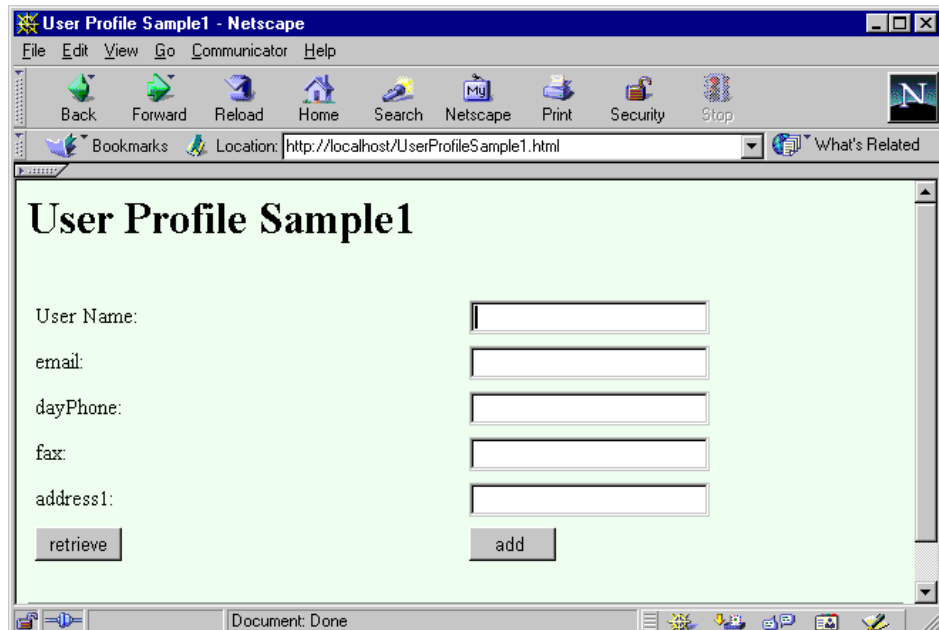


Figure 243. The Referred `UserProfilesample1.html`

5. Enter your data in each of the fields and click the **add** button as shown in Figure 244:

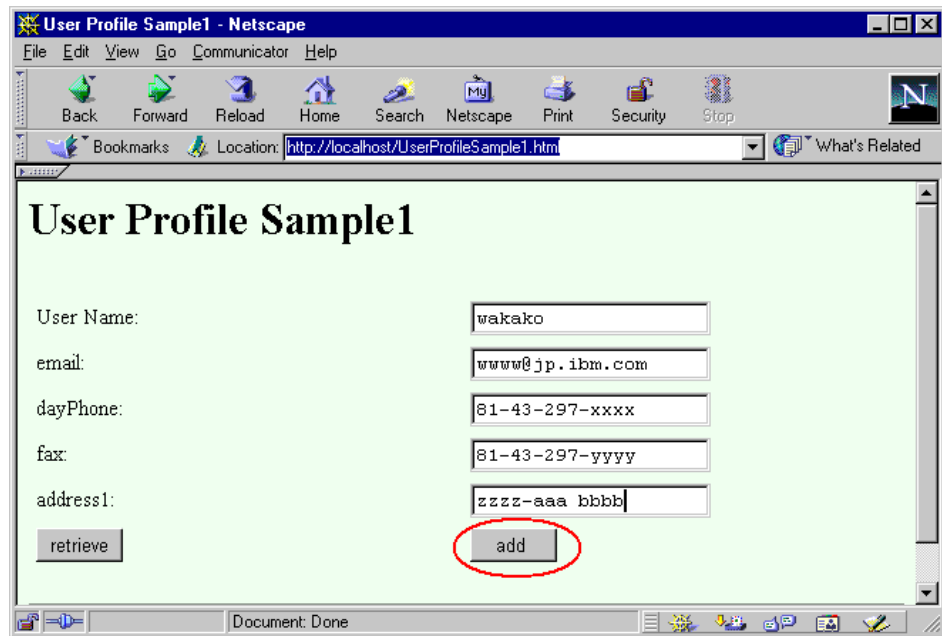


Figure 244. Input Data for Each Field

6. The request will be sent to the UserProfileSample1 servlet, and the result will be returned. It may take a little bit of time on your first request to UserProfile, because the servlet needs to connect to the database and create a new table.

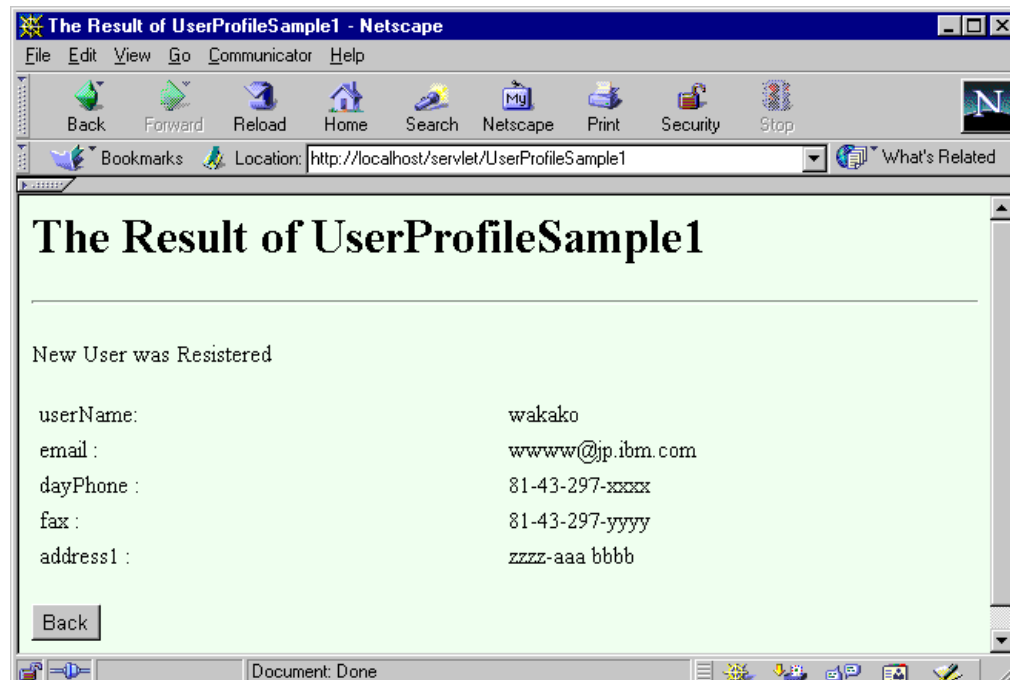


Figure 245. The Result from UserProfile Sample1

7. Click the **Back** button on the page, which brings you back to the UserProfileSample1.html.

8. On the first page, input the name you registered to the userName field and click the **retrieve** button as shown in Figure 244 on page 261.

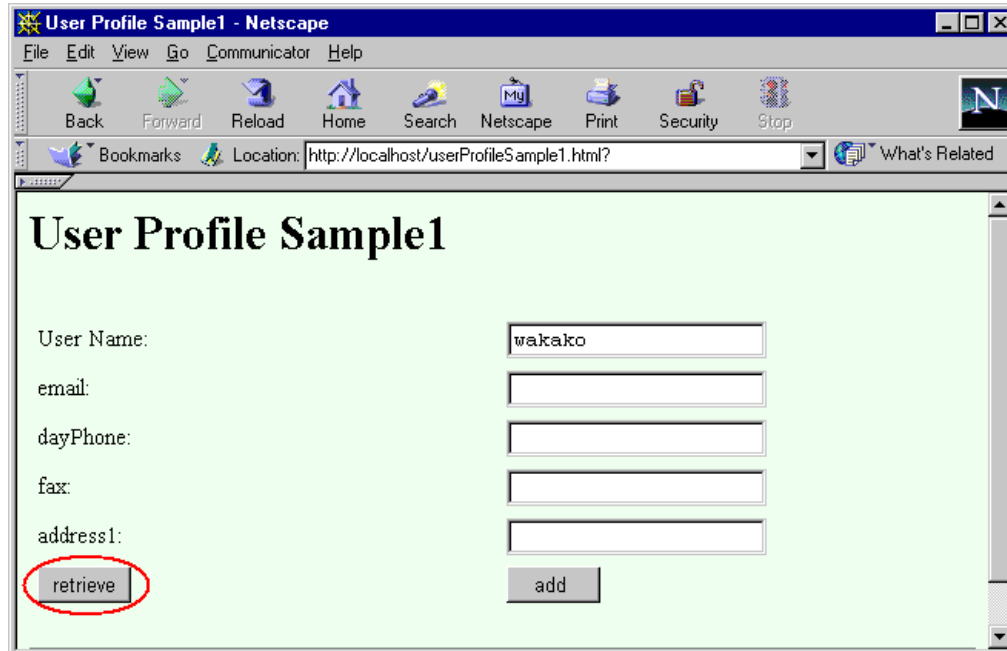


Figure 246. Input the Name You Registered

9. You can get the data that you input earlier as shown in Figure 247:

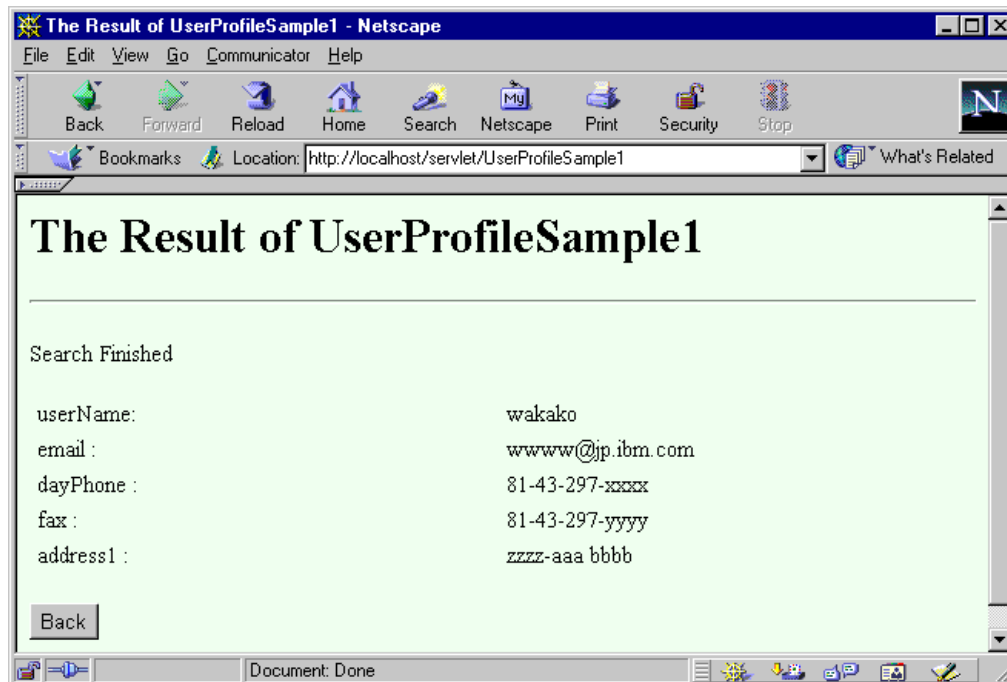


Figure 247. The Retrieved Data

10. If you specify a name that is not registered in the database, the following window will appear:

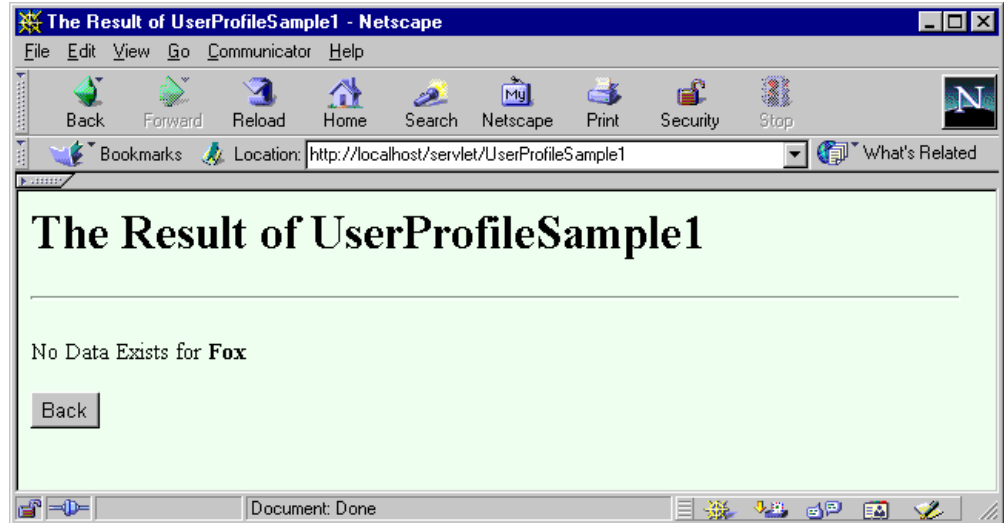


Figure 248. The Result When You Specify Wrong Name

After you work on this sample you will find the table (the default name is `userprofile`) in the database that you specified at 5.2.1, "Setting Up User Profiles" on page 253. You will also find that the Connection Manager was used.

1. To make sure the table was created by the `UserProfile`, if you use DB2 for Windows NT, start the DB2 Control Center by clicking **Start -> Program Files -> DB2 for Windows NT -> Administration Tools -> Control Center** from the Start Menu of Windows NT.
2. The following window will come up (Figure 249). On the left frame of this window, expand the machines node until you find the tables icon in the database name that you specified.
3. Select the **Tables** icon. You will find the user profile table in the right frame.

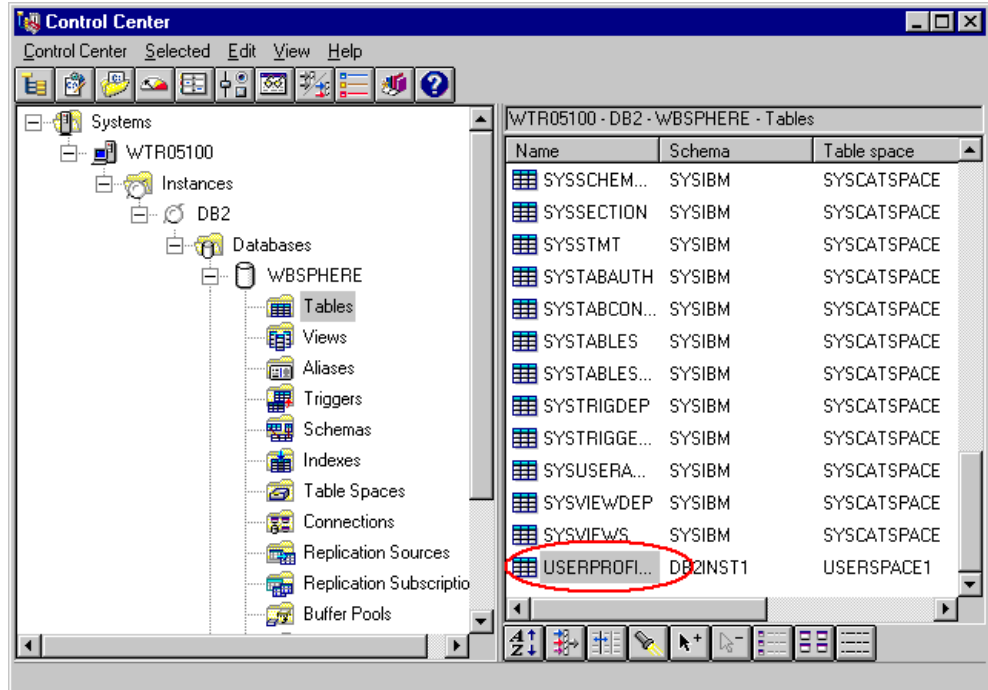


Figure 249. DB2 Control Center

4. Double-click the **USERPROFILE** table icon. This will bring up the following window:

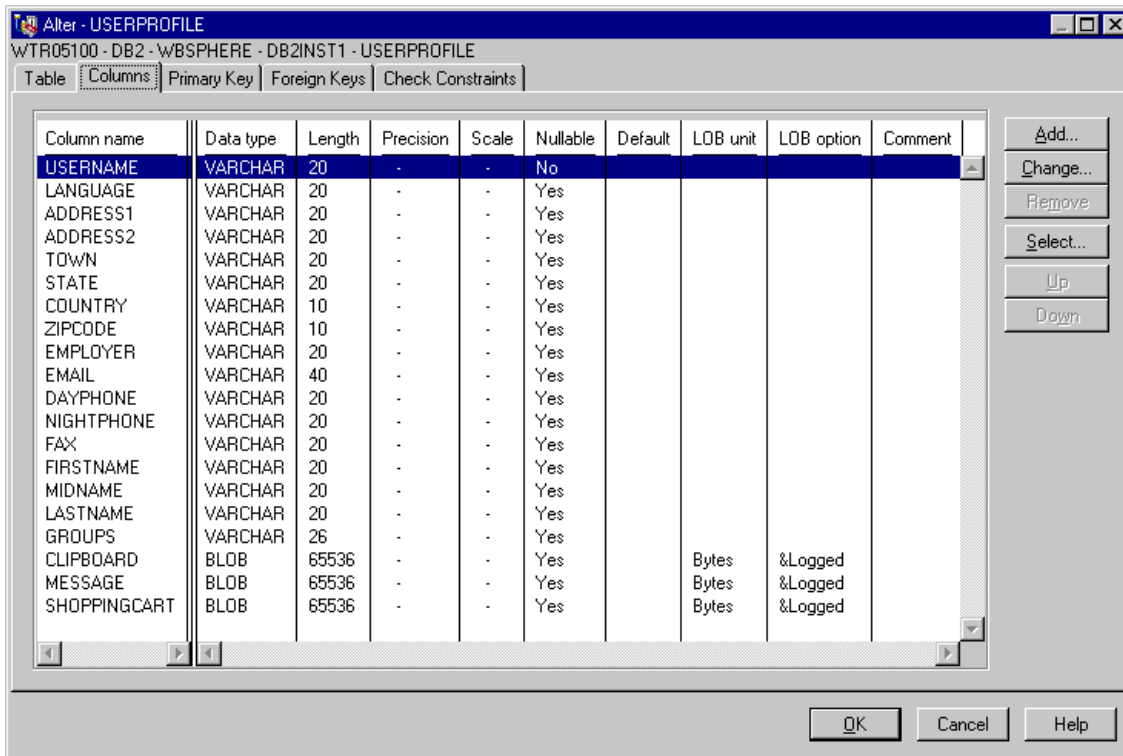


Figure 250. Created User Profile Table

To make sure that the Connection Manager was used:

Click **Server Execution Analysis -> Monitors -> DB Pool Connections** from the Application Server Manager. The following window will come up.

This page to monitor connection pool performance.

Date Interval: 10 seconds Next Update: [Refresh]

Select Pool to View: JdbcDb2

Owner Class	In-Use	TTL	Verify Time	Last Used Time	Connection Type	DB Url	DB User
			21-May-99 3:28:24 PM	21-May-99 3:28:24 PM	JDBC	dbc:db2:wbsphere	db2inst1

Start View
Top view
Refresh
Clear

Pool Name	-	JdbcDb2	Pool Type	-	JDBC	Start Time	-	21-May-99 9:45:57 A
Max connections	-	10	Connect timeout	-	3:20.000	Requests	-	7
Min connections	-	2	Reap time	-	1:00	Waiting	-	0
Total connections	-	7	Max Age	-	:10	Rejected	-	0
Active connections	-	0	Max Idle time	-	:30	Orphaned	-	0
Inactive connections	-	1	Idled	-	0			

Figure 251. UserProfile Uses Connection Manager

5.2.4 Linking User Profiles to Sessions

As mentioned in 5.1.1.2, “HttpSession Object” on page 223, session objects can’t store data persistently. By linking the session objects to the user profile, the data from the session objects can be stored persistently.

The method `addUserProfile(HttpSession)` enables you to associate UserProfiles and SessionObject very easily. `UserProfileSample2.html` and `UserProfileSample2.java` are the samples you can use to link the user profiles and the sessions objects.

Create UserProfileSample2.html and UserProfileSample2.java as shown in Figures 252, 253, and 254.

```
<html> <head>
<title>User Profile Sample2</title>
</head>

<BODY TEXT="#000000" BGCOLOR="#ddeeff">
<h1>User Profile Sample2</h1>
<p><br>

<form action = "http://localhost/servlet/UserProfileSample2" method =
"POST">
<table COLS=2 >
<tr><td>User Name:</td>
<td><input type = "text" name = "userName"></td></tr>

<tr><td>email:</td>
<td><input type = "text" name = "email"></td></tr>

<tr><td>dayPhone:</td>
<td><input type = "text" name = "dayPhone"></td></tr>

<tr><td>fax:</td>
<td><input type = "text" name = "fax"></td></tr>

<tr><td>address1:</td>
<td><input type = "text" name = "address1"></td></tr>

<tr><td><INPUT TYPE="submit" NAME="button" VALUE="retrieve" ></td>
<td><INPUT TYPE="submit" NAME="button" VALUE=" add " ></td></tr>
</table>
</form>
<hr>
<!-- hhmts start -->
Last modified: Wed May 26 09:45:31 1999
<!-- hhmts end -->
</body> </html>
```

Figure 252. UserProfileSample2.html

```

import com.ibm.servlet.personalization.sessiontracking.*;
import com.ibm.servlet.personalization.userprofile.*;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class UserProfileSample2 extends HttpServlet {
    public void doPost(javax.servlet.http.HttpServletRequest
req, javax.servlet.http.HttpServletResponse res) throws ServletException,
IOException {
        String button = null;
        String userName = null;
        String message = null;

        UserProfile up = new UserProfile();
        Hashtable userInfo = new Hashtable();

        button = req.getParameterValues("button") [0];
        userName = req.getParameterValues("userName") [0];

        //-----IBMsession-----
        // Step 1: Get the Session object casting to IBMSessionData
        IBMSessionData isession = (IBMSessionData)req.getSession(true);
        // Step 2: Get the session data value
        Integer ival = (Integer) isession.getValue
("UserProfileSample2.counter");
        if (ival == null){
            ival = new Integer (1);
            isession.setUserName(userName);
        }
        else{
            ival = new Integer (ival.intValue () + 1);
        }
        isession.putValue ("UserProfileSample2.counter", ival);
        //-----IBMsession-----

        if (button.equals("retrieve")){

            if((up = UserProfile.retrieveUserProfileByUserName(userName)) ==
null){
                message = "No Data Exists for <B> "+userName+" </B>";
            }else{
                message = "Search Finished";
            }
        }
    }
}

```

Figure 253. UserProfileSample2.java (1/2)

```

    }else{
        if((up = UserProfile.retrieveUserProfileByUserName(userName)) ==
null){
//up = UserProfile.addUserProfile(userName);
up = UserProfile.addUserProfile(isession);
message = "New User was Resistered";
        }else{
message = "Updated User Info";
        }
        userInfo.put("userName",userName);
        userInfo.put("email",req.getParameterValues("email")[0]);
        userInfo.put("dayPhone",req.getParameterValues("dayPhone")[0]);
        userInfo.put("fax",req.getParameterValues("fax")[0]);
        userInfo.put("address1",req.getParameterValues("address1")[0]);

        up.updateUserProfile(userInfo);
    }
    res.setContentType("text/html");
    res.setHeader("Pragma","No-cache");
    res.setHeader("Cache-Control","no-cache");
    res.setDateHeader("Expires",0);

    PrintWriter pw = res.getWriter();

    pw.println("<HTML><HEAD>");
    pw.println("<TITLE>The Result of UserProfileSample2</TITLE></HEAD>");
    pw.println("<BODY TEXT=\\"#000000\\" BGCOLOR=\\"#ddeeff\\">");
    pw.println("<H1>The Result of UserProfileSample2</H1>");
    pw.println("<HR WIDTH=\\"100%\\">");
    pw.println("<BR>" + message + "<BR>");
    pw.println("<B>You have hit this page " + ival + " times" +
"</B><br>");
    pw.println("<FORM ACTION = \\"http://localhost/userProfileSample2.html\\"
METHOD = \\"GET\\">");
    if(up != null){
        pw.println("<table COLS=2 ><B>");

pw.println("<tr><td>userName:</td><td>" +up.getUserName()+"</td></tr>");
        pw.println("<tr><td>email :</td><td>" +up.getEmail()+"</td></tr>");
        pw.println("<tr><td>dayPhone
:</td><td>" +up.getDayPhone()+"</td></tr>");
        pw.println("<tr><td>fax :</td><td>" +up.getFax()+"</td></tr>");
        pw.println("<tr><td>address1 :</td><td>"
+up.getAddress1()+"</td></tr>");
        pw.println("</B></table>");
    }
    pw.println("<P><INPUT TYPE = \\"submit\\" VALUE = \\"Back\\"></FORM>");
    pw.println("</BODY></HTML>");

    pw.flush();
    pw.close();
}
}

```

Figure 254. UserProfileSample2.java (2/2)

1. Copy UserProfileSample2.html into your HTTP document root. For example, the default document root for IBM HTTP Server V1.3.3 on Windows NT is \Program Files\ IBM HTTP Server\htdocs.
2. Copy UserProfileSample2.java into <ASRoot>/servlets. Compile UserProfileSample2.java and create the UserProfileSample2.class file.

```
C:\WEBSPH~1\APPSER~1\servlets>javac UserProfileSample2.java  
C:\WEBSPH~1\APPSER~1\servlets>
```

3. Open the URL `http://<hostname>/UserProfileSample2.html`. The following window will appear:

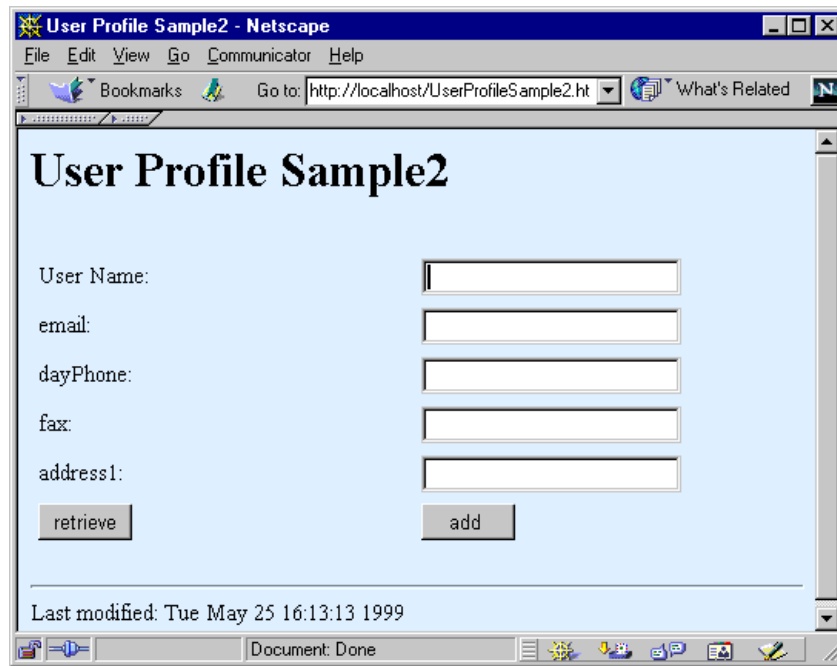


Figure 255. The Referred UserProfileSample2.html

4. On this page, input the name you registered in 5.2.3, “UserProfile Sample” on page 256 to the User Name field and click the **retrieve** button as shown in Figure 256 on page 270.

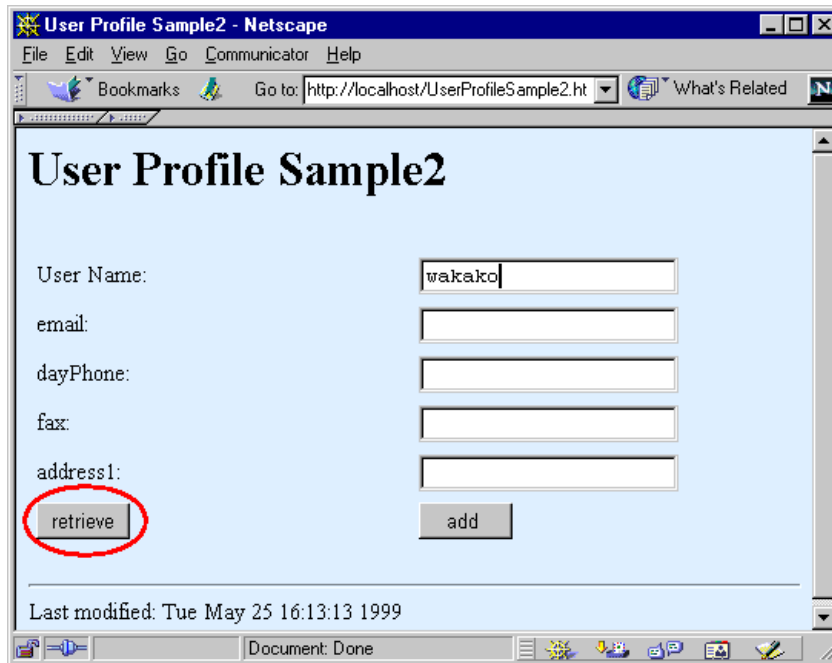


Figure 256. Input the Name You Registered Earlier

5. You can get the data that you input earlier and the value of the counter in the session object as shown in Figure 257:

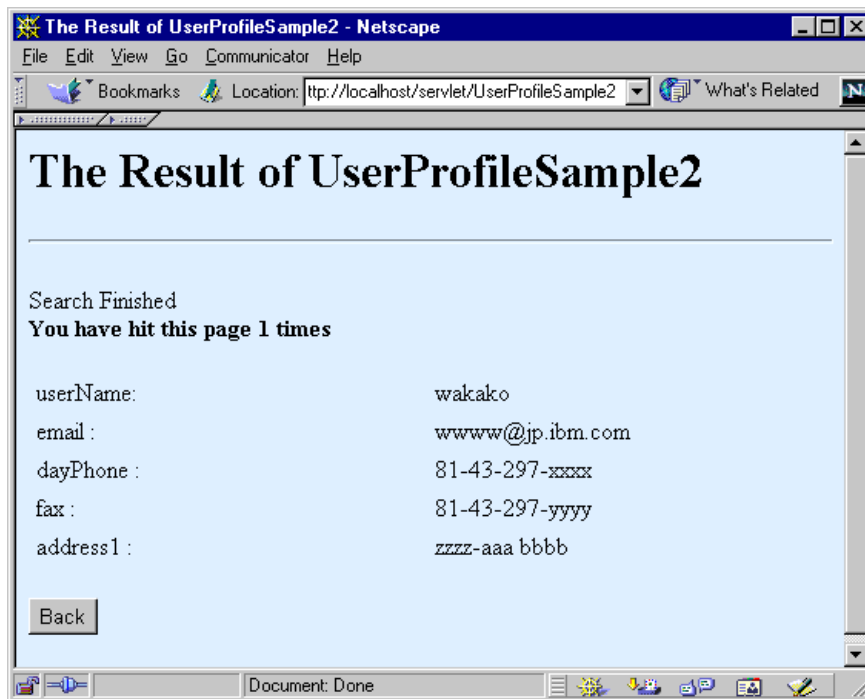


Figure 257. The Retrieved Data

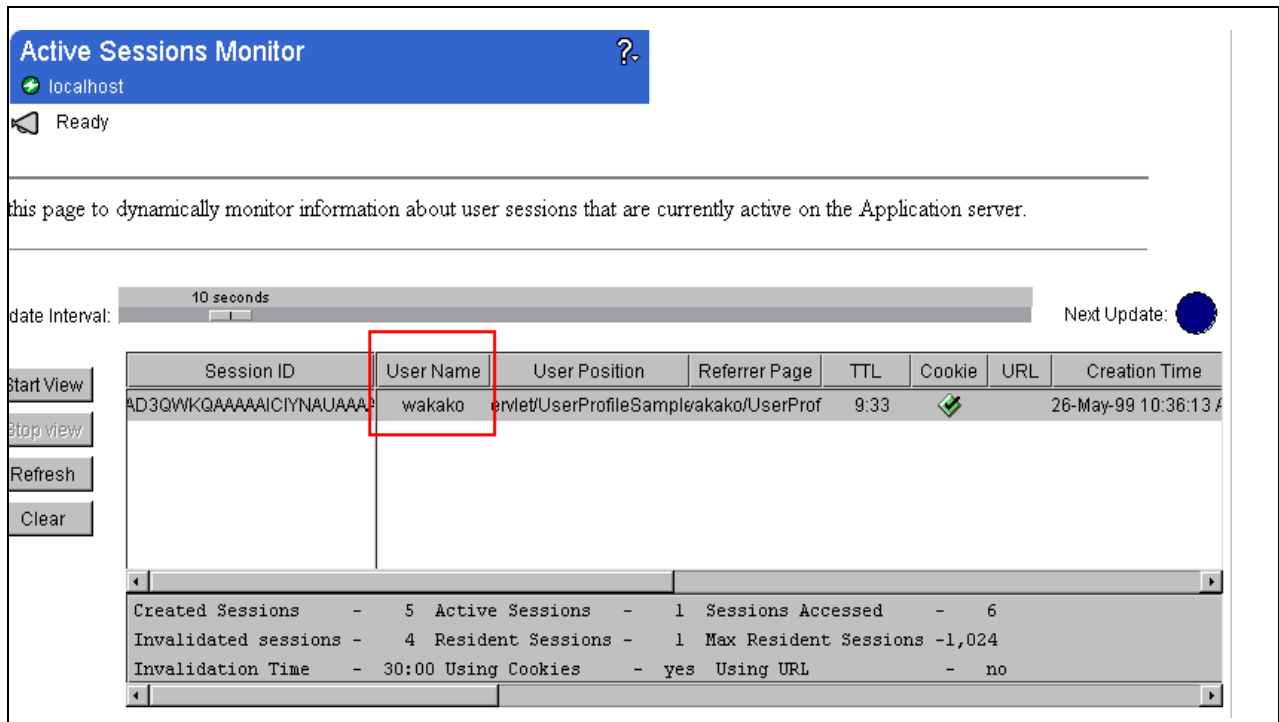


Figure 258. The Session Object Associated with UserProfile

5.2.5 Extending the UserProfile Class

Even though the UserProfile has enough fields to store user data, it is possible that you might need extra fields to meet your application requirements. You can extend your user profile by extending `com.ibm.servlet.personalization.userprofile.UserProfile`. In this part, we show how to configure your Application Server Manager for a new user profile and how to extend the user profile from the servlet.

5.2.5.1 Extending the UserProfile Class

There is a sample extension of the WebSphere UserProfile class, `UserProfile2.java`, in the directory `<ASRoot>\samples\userprofile`. It shows how to add fields to the default user profile class. The source code for `com.ibm.servlet.personalization.userprofile.UserProfile`, `UserProfile.java` is also in the same directory. In `UserProfile2.java`, there are detailed comments to help extend the UserProfile. Therefore, you can just copy `UserProfile2.java` and modify the part that refers to `userprofile.java`. Basically `UserProfile3.java`, which we created for this sample, is almost the same as `UserProfile2.java`. Only the `addUserProfile` method is modified and the `updateUserProfile` method is created for the sample application used in 5.2.5.3, “Using Customized UserProfile” on page 280.

To extend the UserProfile class:

1. Declare data members specifying new columns and the attribute of the column in the database.

```

import java.net.*;
import java.sql.*;
import java.util.*;
import java.io.*;

import javax.servlet.*;
import javax.servlet.http.*;
import com.sun.server.*;
import com.ibm.servlet.personalization.userprofile.UserProfile;
import com.ibm.servlet.personalization.sessiontracking.*;
import com.ibm.servlet.connmgr.*;
import com.ibm.servlet.debug.DebugSupport;

```

Figure 259. *UserProfile3.java (1/9)*

2. Create an SQL statement to add the new column.

```

public class UserProfile3 extends
com.ibm.servlet.personalization.userprofile.UserProfile
{

    static String newColumnName = "cellPhone"; /* the new column */
    static String newColumnType = "varchar (20)";
    private String cellPhone; /* the data member that will store this column */
    static{
        Connection con = null;
        IBMJdbcConn cmConn = null;
        //cmConn = (IBMJdbcConn)cm.getIBMConnection (spec);

        try{
            cmConn = (IBMJdbcConn)connMgr.getIBMConnection (spec,"UserProfile3");
            con = cmConn.getJdbcConnection();
            Statement s = con.createStatement();

            s.executeUpdate("alter table " + tableName + " add " + newColumnName +
newColumnType + " ");
            s.close();
        }catch (SQLException e){
            e.getMessage();
        }catch (IBMConnMgrException e){
            System.out.println (e.getMessage());
        }finally{
            if (con != null){
                try{
                    cmConn.releaseIBMConnection();
                }catch(IBMConnMgrException e2){
                    System.out.println (e2.getMessage());
                }
            } //if
        } //finally
    } //static
}

```

Figure 260. *UserProfile3.java (2/9)*

3. This is a constructor (getter and setter methods) for the new data item (Figure 261).

```
public UserProfile3(){
    super();
}
public String getCellPhone() {
    return cellPhone;
}

public synchronized void setCellPhone(String value){
    { cellPhone = value;}
    updateUPField(newColumnName, value);
}
```

Figure 261. *UserProfile3.java* (3/9)

4. The `retrieveUserProfile3` method is in Figure 264 on page 276. You must provide a new `retrieveUserProfile` method. Modify the parameter for the `executeQuery` method to meet the new data requirement.

```

static protected UserProfile3 retrieveUserProfile3(String type, String
key){
    UserProfile3 result = null;
    result = (UserProfile3)UserProfile.retrieveUserProfile (type, key);/*
first, call the parent's method */

    if (result != null){
        ResultSet rs = null;
        Connection con = null;
        IBMJdbcConn cmConn = null;

        if ( key != null ){
try{
        cmConn = (IBMJdbcConn)connMgr.getIBMConnection (spec);
        con = cmConn.getJdbcConnection();

        Statement s = con.createStatement();
        rs = s.executeQuery("select " + newColumnName + " from " + tableName + "
where " + type + " = '" + key + "'");
        /* Modify the above statement to select all the new data items:
        select newColumnName1, newColumnName2, ... newColumnNamen ..
        */

        while ( rs.next() ){
            result.cellPhone = rs.getString(newColumnName);
        }
        rs.close(); //added
        s.close();
    }catch (SQLException e){
        DebugSupport.logException("invalid UserProfile search criteria:" + type +
" , "+ key, e);
        try{
            if (con != null) con.rollback();
        }catch (SQLException e2){
        }
        return null;
    }
}

```

Figure 262. UserProfile3.java (4/9)

```

}catch (IBMConnMgrException e2){
    System.out.println (e2.getMessage());
}finally{
    try{
        if (rs != null)
            rs.close();
    }catch (SQLException e){
        System.out.println("problem closing result set");
        try{
            if (con != null) con.rollback();
            if (con != null) cmConn.releaseIBMConnection();
        }catch (Exception e2){
            System.out.println (e2.getMessage());
        }
        return result;
    }
}
    }
    if (con != null){
    try{
        cmConn.releaseIBMConnection();
    }catch (IBMConnMgrException e){
        System.out.println (e.getMessage());
    }
    }
    return result;
}
else return null;
}

```

Figure 263. *UserProfile3.java* (5/9)

5. Override `retrieveUserProfileByUserName` (Figure 264 on page 276).
6. Create `retrieveUserProfilesByXXX` where "XXX" is a new data item you created (see Figure 264 on page 276).
7. Override `getUserProfile` to force it to use the subclass's `retrieveUserProfileByUserName` method (Figure 264 on page 276).

```

static public UserProfile retrieveUserProfileByUserName(String userName) {
    return((retrieveUserProfile3("userName",userName)));
}

static public Enumeration retrieveUserProfilesByCellPhone(String cellPhone,
Statement s){
    return(retrieveUserProfiles("cellPhone",cellPhone, s));
}

static public UserProfile getUserProfile(HttpServletRequest req) {
    IBMSessionData ss = (IBMSessionData) (req.getSession(false));
    String userName = req.getRemoteUser();

    if (userName == null)
        userName = UserProfile.ANONYMOUS_USERNAME;
    if ((ss != null) && (ss.isAppUserName()))
        userName = ss.getUserName();

    return(retrieveUserProfileByUserName(userName));
}

```

Figure 264. *UserProfile3.java* (6/9)

8. Override `updateUserProfile(Hashtable clipboard)` to update and store the new data item(s) from the clipboard (Figure 265 on page 277). This method is not in `UserProfile2.java`. We created the method to use this in the sample application shown in 5.2.5.3, “Using Customized UserProfile” on page 280.

9. Override `addUserProfile(Hashtable clipboard)` to retrieve and store the new data item(s) from the clipboard. This method for `UserProfile3` is a little different from `UserProfile2.java`, since `UserProfile3.updateUserProfile(Hashtable clipboard)` is used in this method (see Figure 266 on page 278).

```
static public UserProfile addUserProfile(Hashtable clipboard){
    String v_userName = nullify(clipboard.get ("userName"));

    UserProfile3 userProfile = (UserProfile3)addUserProfile(v_userName);
    if (userProfile == null)
        return null;
    userProfile.updateUserProfile(clipboard);
    return userProfile;
}
```

Figure 266. `UserProfile3.java` (8/9)

10. Override `toHTML` and `toString`:

```
public String toHTML () {
    return(super.toHTML() + "<b> cellPhone</b> : " + cellPhone + "<BR>");
    /* Add code to output any other data items to the string above */
}

public String toString(){
    return(super.toString() + " cellPhone= " + getCellPhone());
    /* Add code to output any other data items to the string above */
}
}
```

Figure 267. `UserProfile3.java` (9/9)

Compile `UserProfile3.java` and create `UserProfile3.class`.

```
C:\WebSphere\AppServer\classes>javac UserProfile3.java
C:\WebSphere\AppServer\classes>
```

Configuring the Application Server Manager for `UserProfile3` is discussed in the next section.

5.2.5.2 Configuring a Customized UserProfile Class

In this section, `UserProfile3` is used as an example of an extended `UserProfile`.

To use `UserProfile3` instead of `com.ibm.servlet.personalization.userprofile.UserProfile`, follow these steps:

1. Copy `UserProfile3.class` into the `<ASRoot>/classes` directory. This directory must be added to the Application Server classpath field in the Java Engine Page of the Application Server Manager (Figure 268 on page 279).

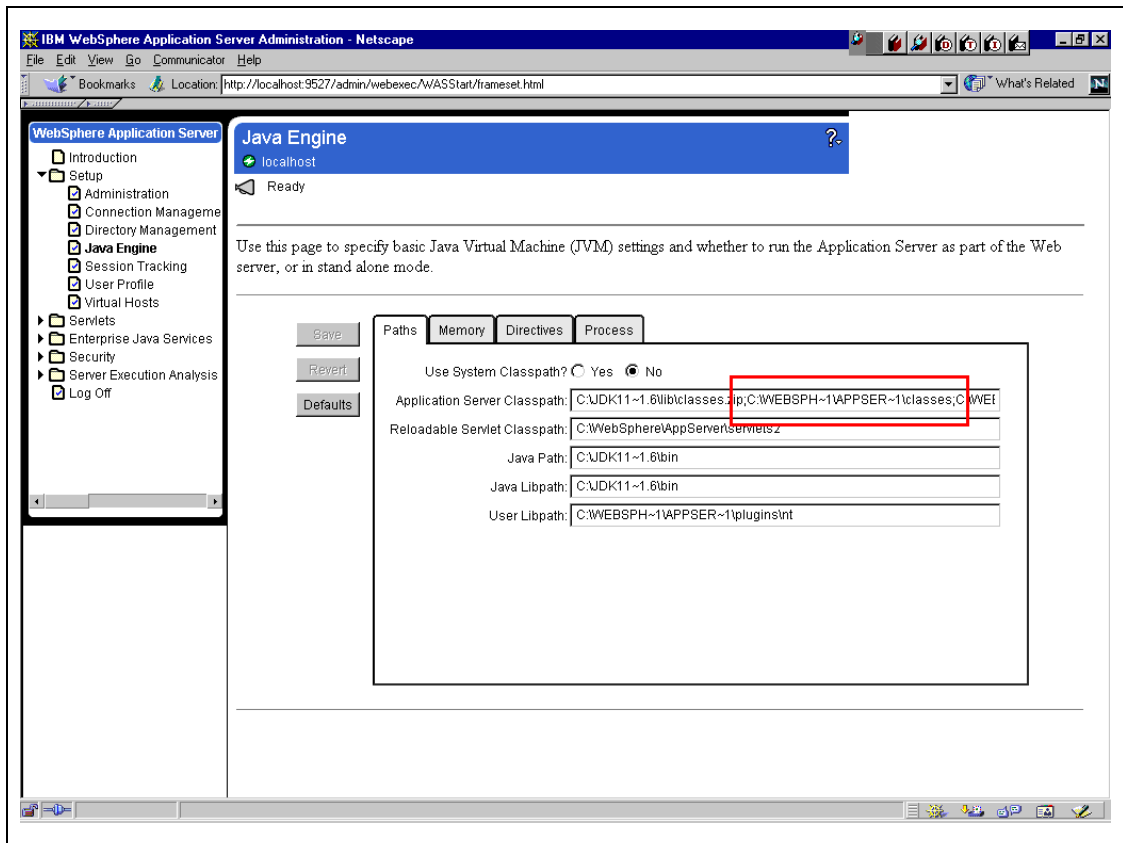


Figure 268. Whatever Directory Is Chosen in the Application Server Classpath Field

2. Open the UserProfile page on the Application Server Manager. Specify your UserProfile class name in the Class Name field on the Enable tab (Figure 269):

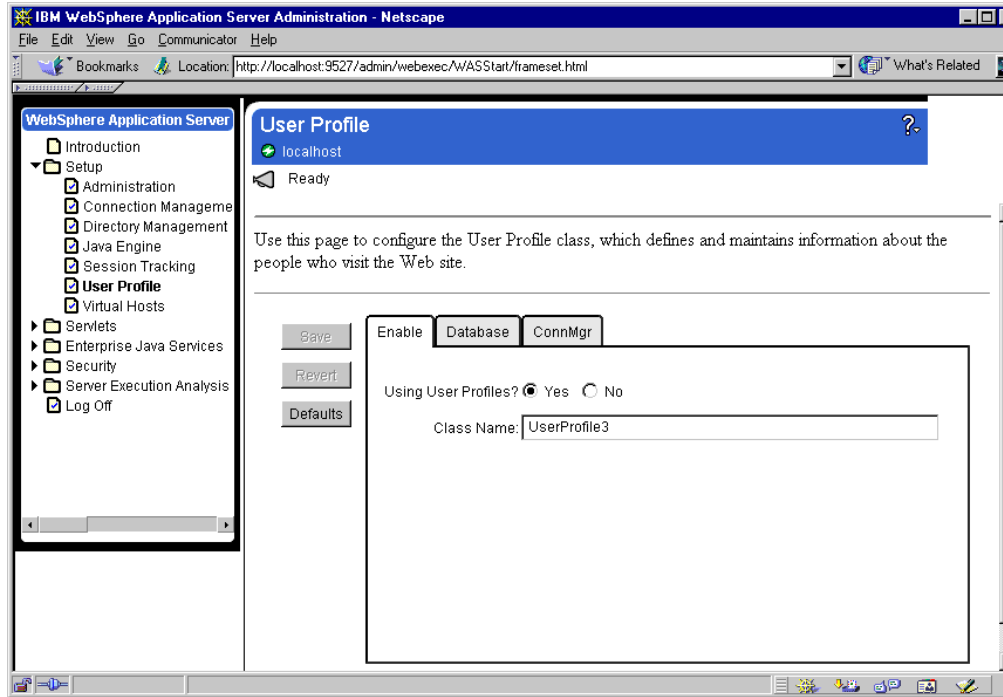


Figure 269. Specify the UserProfile Class Name

3. Click the **Save** button.
4. If the UserProfile is used by any servlet, restart the Application Server. If not, the older UserProfile gets used.

5.2.5.3 Using Customized UserProfile

To use a customized UserProfile, the servlet needs some modifications for the new class.

Figure 270 on page 281, Figure 271 on page 282 and Figure 272 on page 283 (UserProfileSample3.java) are the examples to use for UserProfile3. The bold parts are modified in UserProfile3. Import UserProfile3 instead of `com.ibm.servlet.personalization.userprofile.*`. Use the UserProfile3 method to retrieve and add UserProfile3. Some methods need to point to UserProfile3.

```

import com.ibm.servlet.personalization.sessiontracking.*;
//import com.ibm.servlet.personalization.userprofile.*;
import UserProfile3;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class UserProfileSample3 extends HttpServlet {
    public void doPost(javax.servlet.http.HttpServletRequest
req,javax.servlet.http.HttpServletResponse res) throws ServletException,
IOException {
        String button = null;
        String userName = null;
        String message = null;
//        UserProfile up = new UserProfile();
        UserProfile3 up = new UserProfile3 ();
        Hashtable userInfo = new Hashtable();

        button = req.getParameterValues("button")[0];
        userName = req.getParameterValues("userName")[0];
//-----IBMSession-----
        // Step 1: Get the Session object casting to IBMSessionData
        IBMSessionData isession =(IBMSessionData)req.getSession(true);
        // Step 2: Get the session data value
        Integer ival = (Integer) isession.getValue
("UserProfileSample3.counter");
        if (ival == null){
            ival = new Integer (1);
            isession.setUserName(userName);
        }
        else{
            ival = new Integer (ival.intValue () + 1);
        }
        isession.putValue ("UserProfileSample3.counter", ival);
//-----IBMSession-----
        if (button.equals("retrieve")){

            if((up
=UserProfile3)UserProfile3.retrieveUserProfileByUserName(userName))
            == null){
                message = "No Data Exists for <B> "+userName+" </B>";
            }else{
                message = "Search Finished";
            }
        }
    }
}

```

Figure 270. UserProfileSample3.java (1/3)

```

}else{
    userInfo.put("userName", userName);
    userInfo.put("email", req.getParameterValues("email")[0]);
    userInfo.put("dayPhone", req.getParameterValues("dayPhone")[0]);
    userInfo.put("fax", req.getParameterValues("fax")[0]);
    userInfo.put("address1", req.getParameterValues("address1")[0]);
    userInfo.put("cellPhone", req.getParameterValues("cellPhone")[0]);
    if((up =
(UserProfile3)UserProfile3.retrieveUserProfileByUserName(userName)) ==
null){
up = (UserProfile3)UserProfile3.addUserProfile(userName);
message = "New User was Resistered";
    }else{
message = "Updated User Info";
    }

    up.updateUserProfile(userInfo);
}
res.setContentType("text/html");
res.setHeader("Pragma", "No-cache");
res.setHeader("Cache-Control", "no-cache");
res.setDateHeader("Expires", 0);

PrintWriter pw = res.getWriter();

pw.println("<HTML><HEAD>");
pw.println("<TITLE>The Result of UserProfileSample3</TITLE></HEAD>");
pw.println("<BODY TEXT=\"#000000\" BGCOLOR=\"#ffeedd\">");
pw.println("<H1>The Result of UserProfileSample3</H1>");
pw.println("<HR WIDTH=\"100%\">");
pw.println("<BR>" + message + "<BR>");
pw.println("<B>You have hit this page " + ival + " times" +
"</B><br>");
pw.println("<FORM ACTION = \"http://localhost/userProfileSample3.html\"
METHOD = \"GET\">");

```

Figure 271. UserProfileSample3.java (2/3)

```

    up
    =(UserProfile3)UserProfile3.retrieveUserProfileByUserName(userName);
    if(up != null){
        pw.println("<table COLS=2 ><B>");

pw.println("<tr><td>userName:</td><td>" +up.getUserName()+"</td></tr>");
        pw.println("<tr><td>email :</td><td>" +up.getEmail()+"</td></tr>");
        pw.println("<tr><td>dayPhone
: </td><td>" +up.getDayPhone()+"</td></tr>");
        pw.println("<tr><td>cellPhone
: </td><td>" +up.getCellPhone()+"</td></tr>");
        pw.println("<tr><td>fax :</td><td>" +up.getFax()+"</td></tr>");
        pw.println("<tr><td>address1 :</td><td>"
+up.getAddress1()+"</td></tr>");
        pw.println("</B></table>");
    }
    pw.println("<P><INPUT TYPE = \"submit\" VALUE = \"Back\"></FORM>");
    pw.println("</BODY></HTML>");

    pw.flush();
    pw.close();
}
}

```

Figure 272. UserProfileSample3.java (3/3)

```

<html> <head>
<title>User Profile Sample3</title>
</head>

<BODY TEXT="#000000" BGCOLOR="#ffeedd">
<h1>User Profile Sample3</h1><p><br>

<form action = "http://localhost/servlet/UserProfileSample3" method =
"POST">
<table COLS=2 >
<tr><td>User Name:</td>
<td><input type = "text" name = "userName"></td></tr>

<tr><td>email:</td>
<td><input type = "text" name = "email"></td></tr>

<tr><td>dayPhone:</td>
<td><input type = "text" name = "dayPhone"></td></tr>

<tr><td>cellPhone:</td>
<td><input type = "text" name = "cellPhone"></td></tr>

<tr><td>fax:</td>
<td><input type = "text" name = "fax"></td></tr>

<tr><td>address1:</td>
<td><input type = "text" name = "address1"></td></tr>

<tr><td><INPUT TYPE="submit" NAME="button" VALUE="retrieve" ></td>
<td><INPUT TYPE="submit" NAME="button" VALUE=" add " ></td></tr>
</table>
</form>
<hr>
<!-- hhmts start -->
Last modified: Mon May 31 17:48:40 1999
<!-- hhmts end -->
</body> </html>

```

Figure 273. *UserProfileSample3.html*

To use this sample:

1. Copy *UserProfileSample3.html* into your HTTP document root. For example, the default document root for IBM HTTP Server V1.3.3 on Windows NT is `\Program Files\ IBM HTTP Server\htdocs`.
2. Copy *UserProfileSample3.java* into `<ASRoot>/servlets`. Compile *UserProfileSample3.java* and create the *UserProfileSample3.class* file.

```

C:\WEBSPH~1\APPSER~1\servlets>javac UserProfileSample3.java
C:\WEBSPH~1\APPSER~1\servlets>

```

3. Open the URL `http://<hostname>/UserProfileSample3.html`. The following window will appear. Specify each of the fields and click the **add** button (Figure 274):

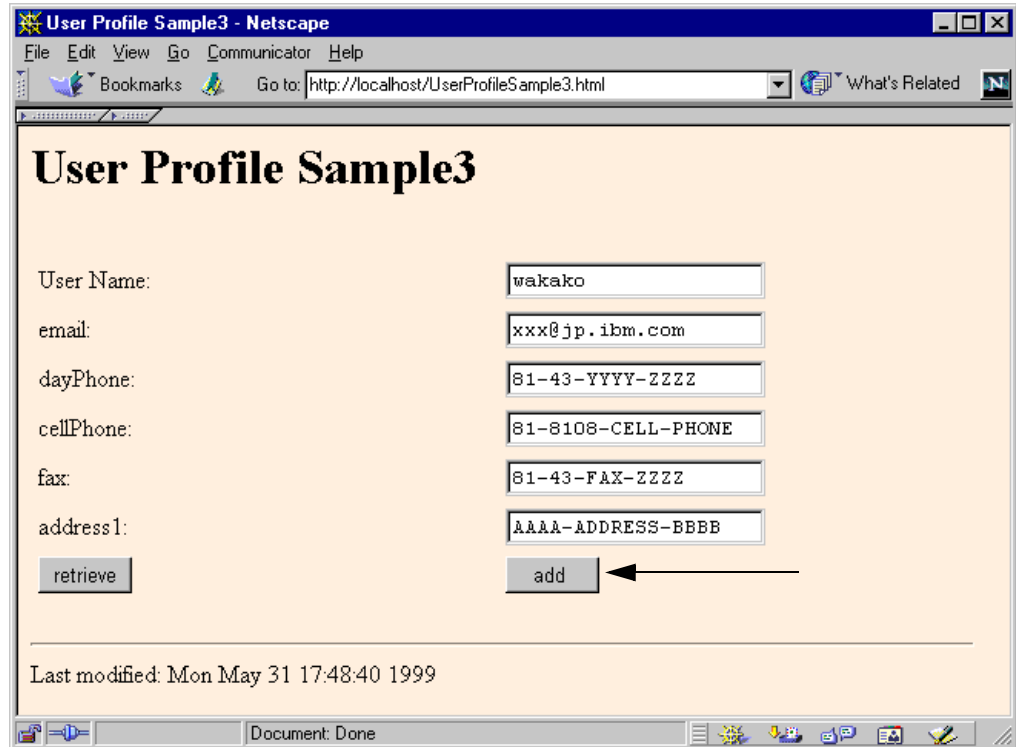


Figure 274. Specify Each of the Fields and Click on the add Button

4. In the following window, you can see the updated value of the cellPhone field:

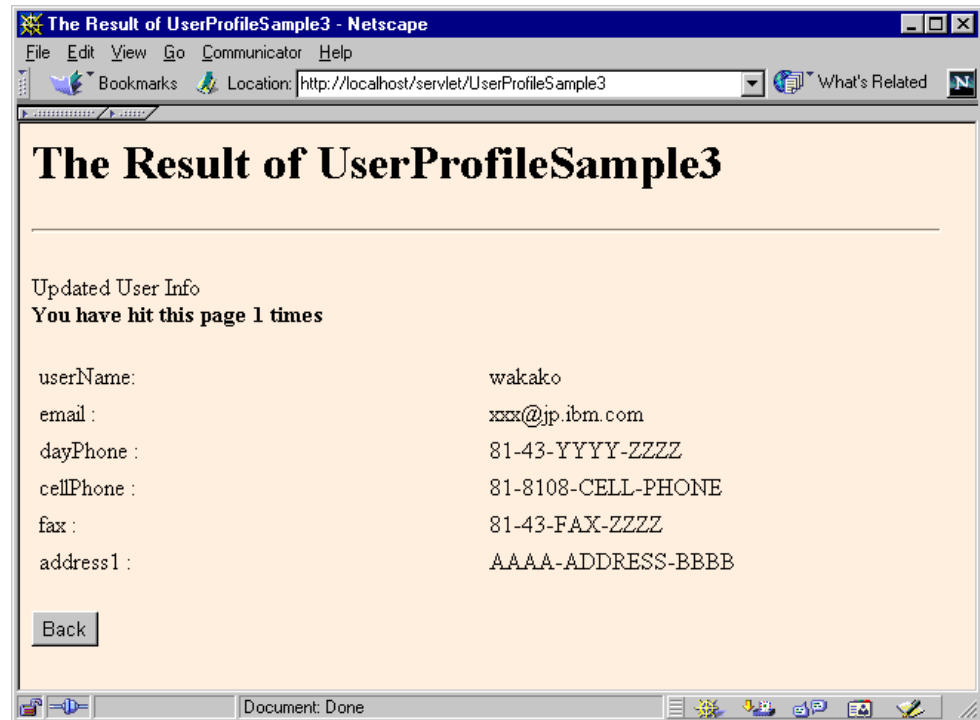


Figure 275. The cellPhone Field Was Updated as well as Other Fields

5.3 Using the Personalization Utilities

The package `com.ibm.servlet.servlets.personalization.util` contains servlets that enable Web administrators to post site-wide bulletins and also permits Web visitors to exchange messages. These servlets' class files are copied to the servlet directory and are registered to the Servlet Configuration page when you install WebSphere Application Server. Therefore, you can use these utilities with some minor additional configuration.

5.3.1 Creating Bulletins

`SetVariableText` and `GetVariableText` in the package `com.ibm.servlet.servlets.personalization.util` are useful servlets to add bulletins or news flashes to your Web site on a page-by-page basis.

You can:

- Store the message for the Site Queue with `SetVariableText` servlet.
- Embed the message from the Site Queue to your Web page with `GetVariableText` instance.

5.3.1.1 How to Store a Message in the Site Queue

To make a site queue and store a message, you can use the `SetVariableText` servlet. This servlet has to be used with the `<SERVLET>` tag in a JSP file as shown in Figure 276. That provides access to the advertiser to store messages.

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html">
  <title>Create Bulletin</title>
</head>
<body>
<H1>Create Bulletin</H1>

<SERVLET NAME="SetVariableText">
</SERVLET>
<HR>
</body>
</html>
```

Figure 276. `CreateBulletin.jsp`

1. Create the `CreateBulletin.jsp` file and place it in your HTTP document root. For example, the default document root of IBM HTTP Server V1.3.3 on Windows NT is `\Program Files\IBM HTTP Server\htdocs`.
2. Open the URL `http://<hostname>/CreateBulletin.jsp`. The following window will appear:

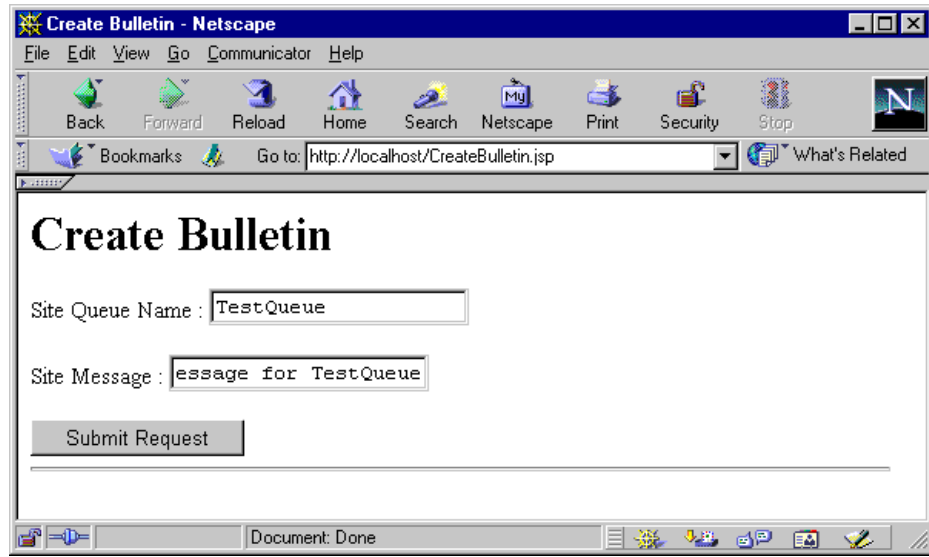


Figure 277. Set the Site Queue Name via `SetVariableText`

3. Specify the site queue name and type your message in the Site Message field.
4. Click **Submit Request** button and the following window appears:

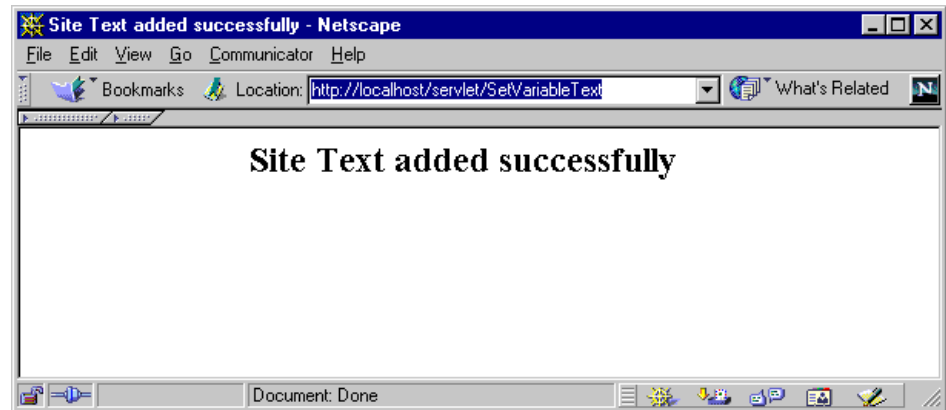


Figure 278. The Message Added Successfully

5.3.1.2 Showing Bulletin

To propagate the message created in the previous steps:

1. Add the `<SERVLET>` tags with `GetVariableText` and `queueName` properties to the Web page on which you want to show the message, for example, `Bulletin.jsp` (Figure 279):

```

<html>
<head>
  <meta http-equiv="Content-Type" content="text/html">
  <title>Bulletin Test</title>
</head>
<body>
<H1>Bulletin Test1 (TestQueue)</H1>
<H2>
<SERVLET NAME="GetVariableText" queueName="TestQueue">
</SERVLET>
</H2>
</body>
</html>

```

Figure 279. *Bulletin.jsp*

2. After placing *Bulletin.jsp* in your HTTP document root, open the URL `http://<hostname>//Bulletin.jsp`. You can see the message stored in *TestQueue* (Figure 280 on page 288).

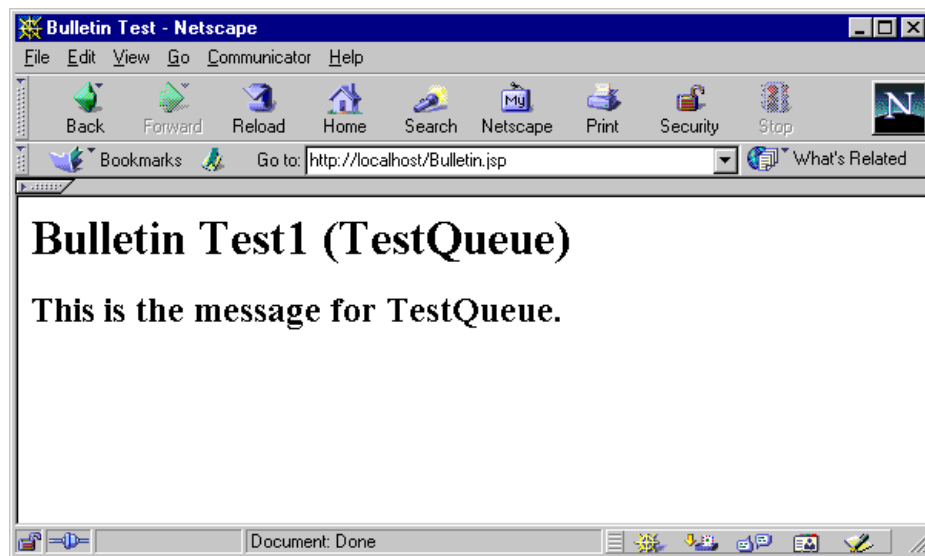


Figure 280. *Stored Message from TestQueue*

5.3.1.3 Creating Multiple Site Queues in a Server

When you want to create multiple site queues in a server (which is a common requirement), you need additional configurations for the instance of *GetVariableText* servlet. Since the *queueName* parameter is passed to the servlet as an initial parameter, once the instance of the *GetVariableText* servlet is loaded, the parameter can't be changed until the servlet is unloaded. For example, if you store different messages in the *TestQueue* and the plants' queues (Figure 282), *GetVariableText* can't handle both these queues at once (Figure 283 and Figure 284 on page 290). You need to make a new instance of *GetVariableText* for each of the new site queues.

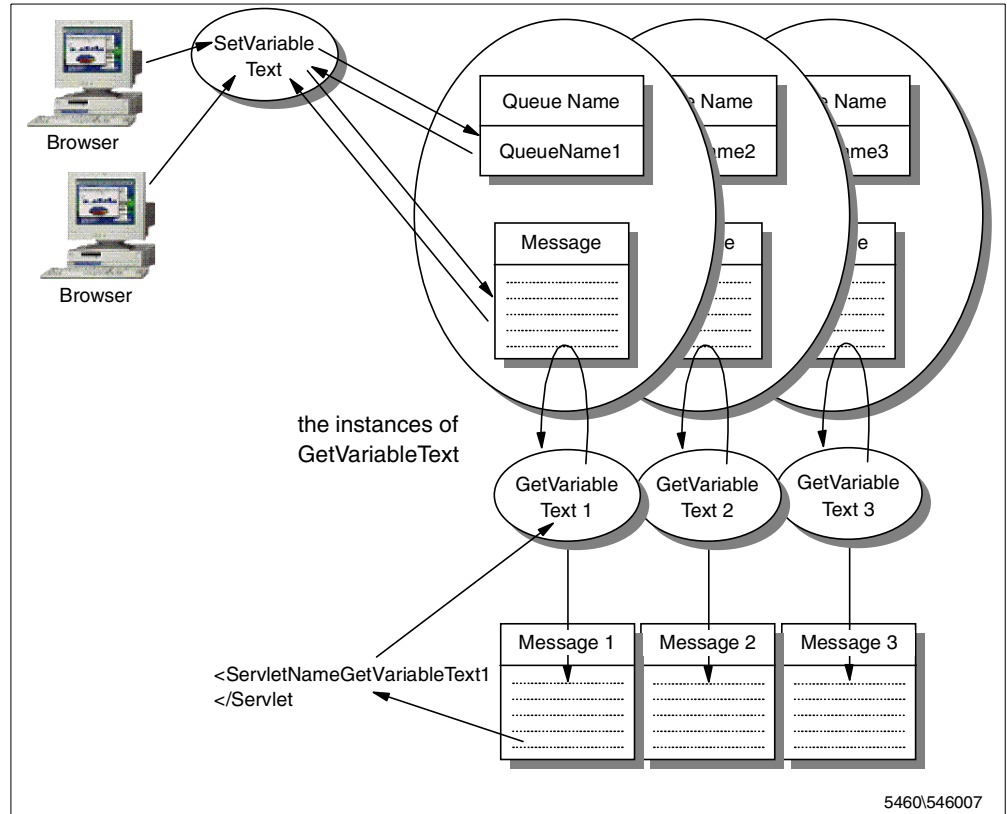


Figure 281. GetVariableText

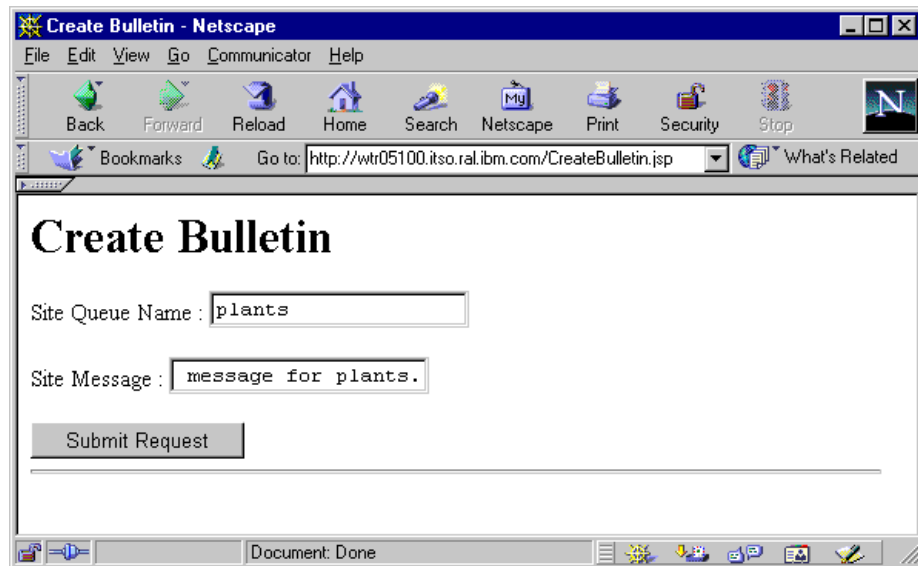


Figure 282. If You Store the Different Messages to the Plants Queue

```

<html><head>
  <meta http-equiv="Content-Type" content="text/html">
  <title>Two Site Queues 1(TestQueue, plants)</title>
</head><body>
<H1>Two Site Queues 1(TestQueue, plants)</H1><HR>
This is the message of the TestQueue.
<H2>
<SERVLET NAME="GetVariableText" queueName="TestQueue">
</SERVLET>
</H2><HR>
Is this the message of the plants.....<B?</B>
<H2>
<SERVLET NAME="GetVariableText" queueName=plants>
</SERVLET>
</H2><HR>
</body></html>

```

Figure 283. TwoSiteQueues1.jsp

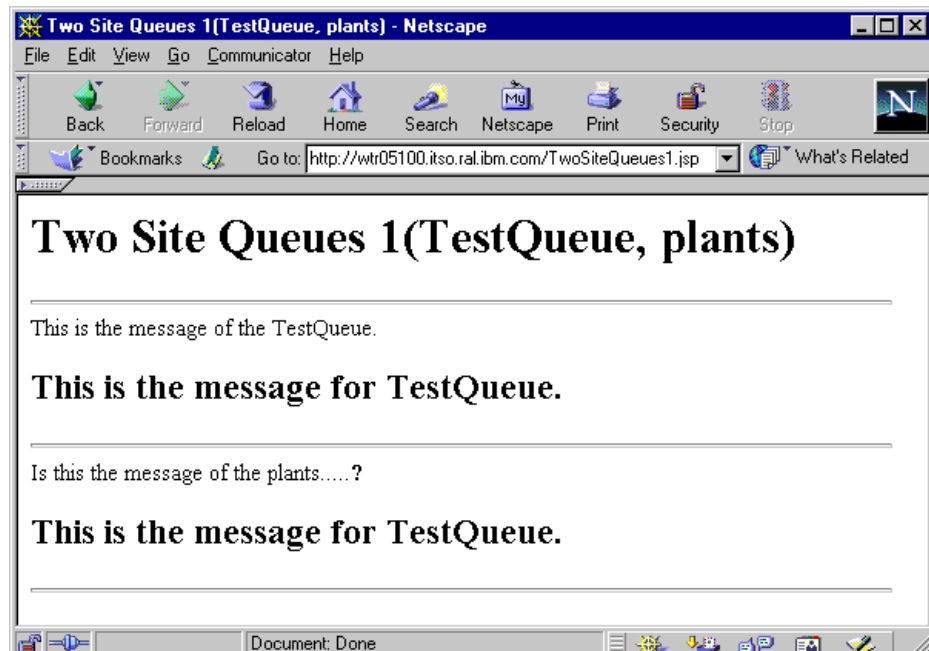


Figure 284. The Referred TwoSiteQueues1.jsp

To create another servlet instance:

1. Go to the Servlet Configuration page by clicking **Servlets -> Configuration** from the Application Server Manager. You can see the GetVariableText servlet is already registered to this page as shown in Figure 285 on page 291.

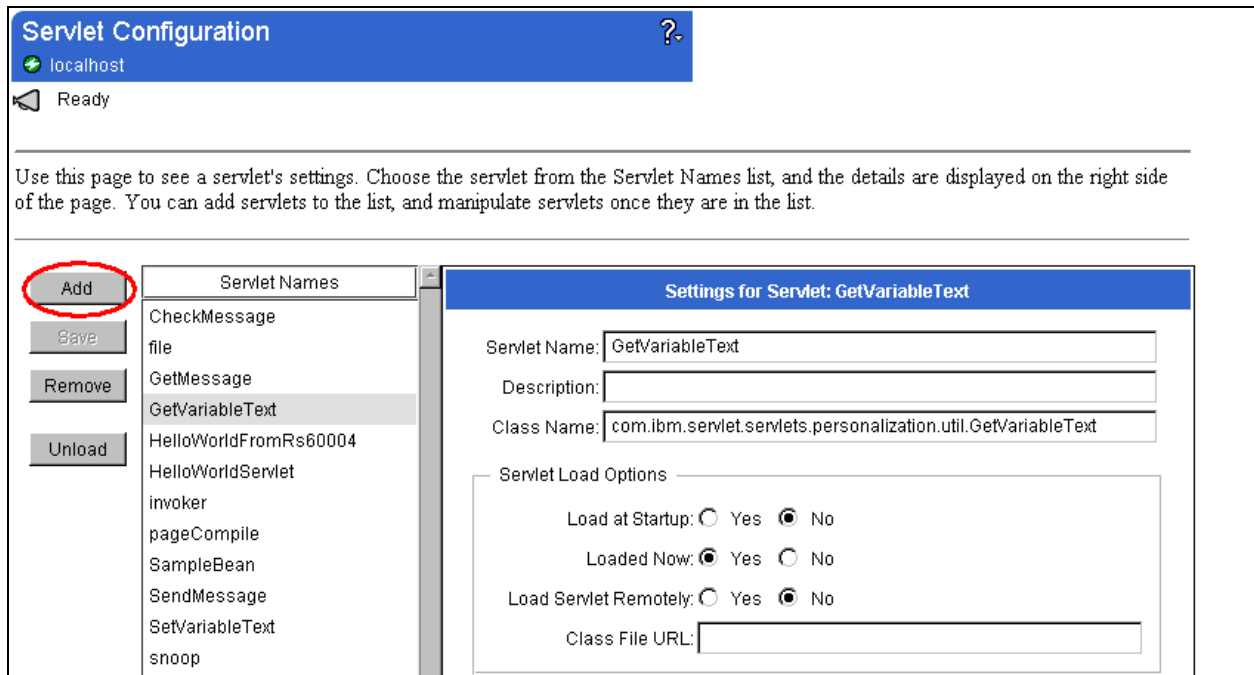


Figure 285. Servlet Configuration Page

Click the **Add** button on this page and add “PlantGetVarText”. Place `com.ibm.servlet.servlets.personalization.util.GetVariableText` in the Servlet Class field. Click the **Add** and **Save** buttons (Figure 286).

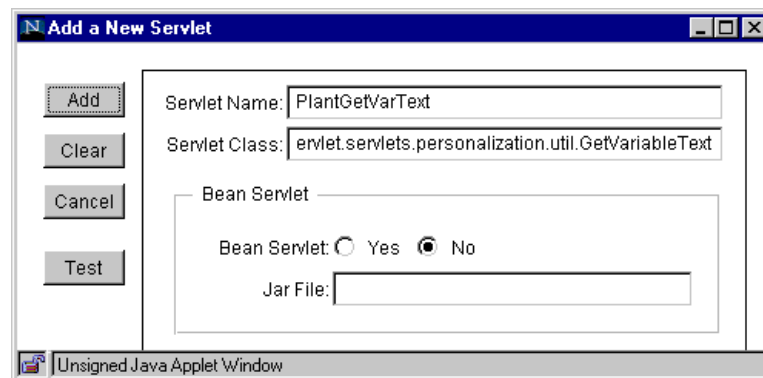


Figure 286. Add a PlantGetVarText Servlet

Place the `TwoSiteQueues2.jsp` (Figure 287) in your HTTP document root.

```

<html><head>
  <meta http-equiv="Content-Type" content="text/html">
  <title>Two Site Queues 2(TestQueue, plants)</title>
</head><body>
<H1>Two Site Queues 2(TestQueue, plants)</H1><HR>
This is the message of the TestQueue.
<H2>
<SERVLET NAME="GetVariableText" queueName="TestQueue">
</SERVLET>
</H2><HR>
This is the message of the plants.
<H2>
<SERVLET NAME="PlantGetVarText" queueName=plants>
</SERVLET>
</H2><HR>
</body></html>

```

Figure 287. TwoSiteQueues2.jsp

2. Open the URL `http://<hostname>/TwoSiteQueues2.jsp`. You can see the messages for each of the queues at this time (Figure 288):

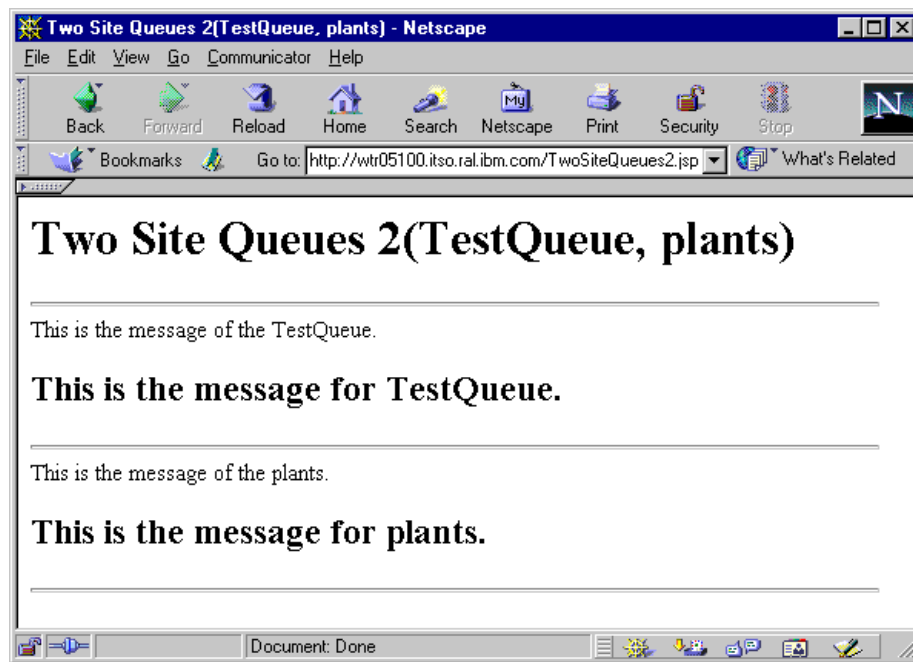


Figure 288. The Referred TwoSiteQueues2.jsp

You can specify the `queueName` property on the Servlet Configuration page and in the `<SERVLET>` tags.

1. Click the **Add** button in the Servlet Properties field and specify the `queueName` property name and value (Figure 289 on page 293). Click **Save**.

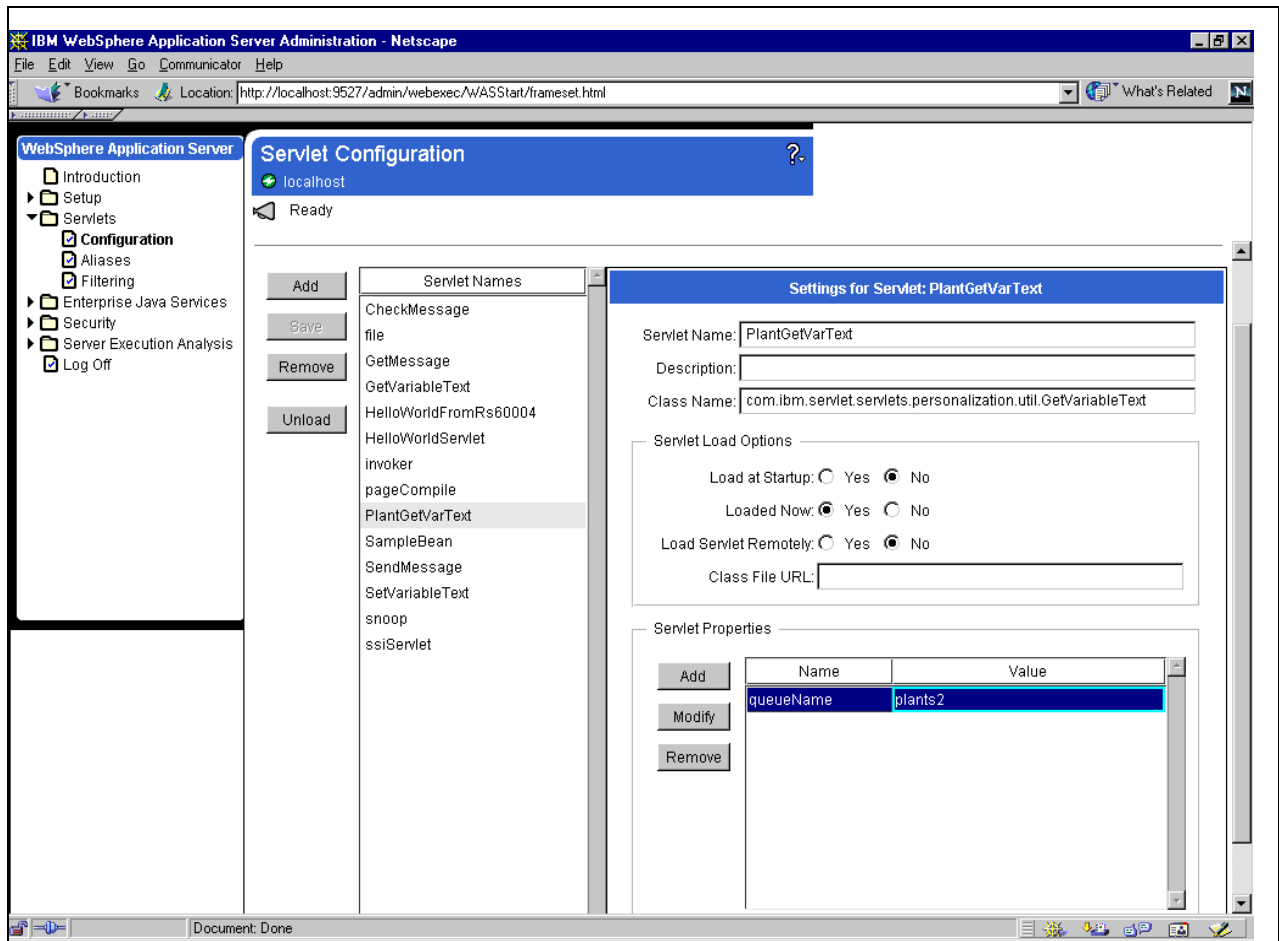


Figure 289. Specifying queueName Property on Servlet Configuration Page

2. Store the message in the plants2 site queue (Figure 290):

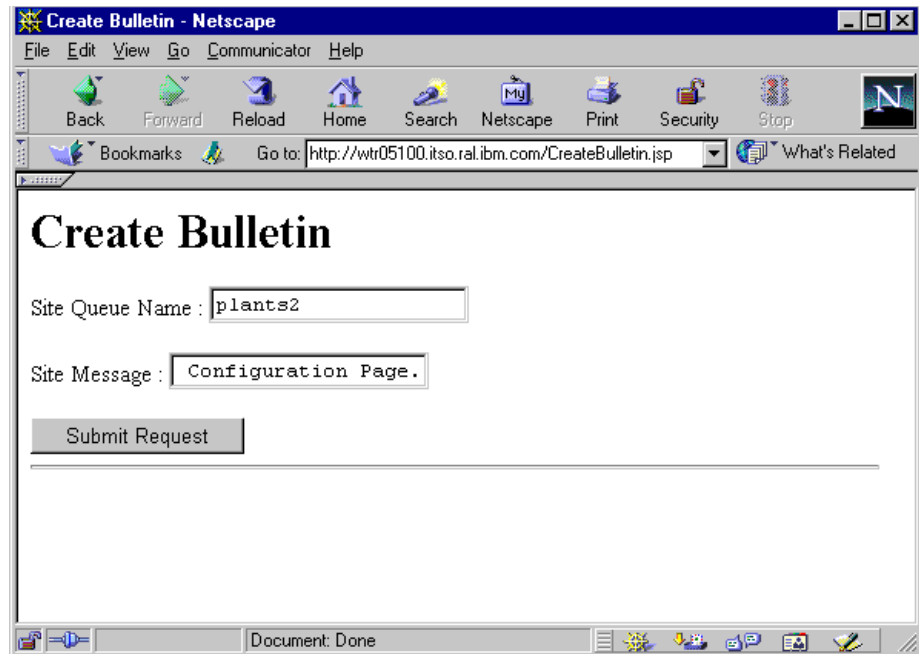


Figure 290. Storing the Message to plants2 Queue

3. Open the URL `http://<hostname>/TwoSiteQueues2.jsp` and the following window will appear. The `queueName` specified in the `<SERVLET>` tag is ignored:

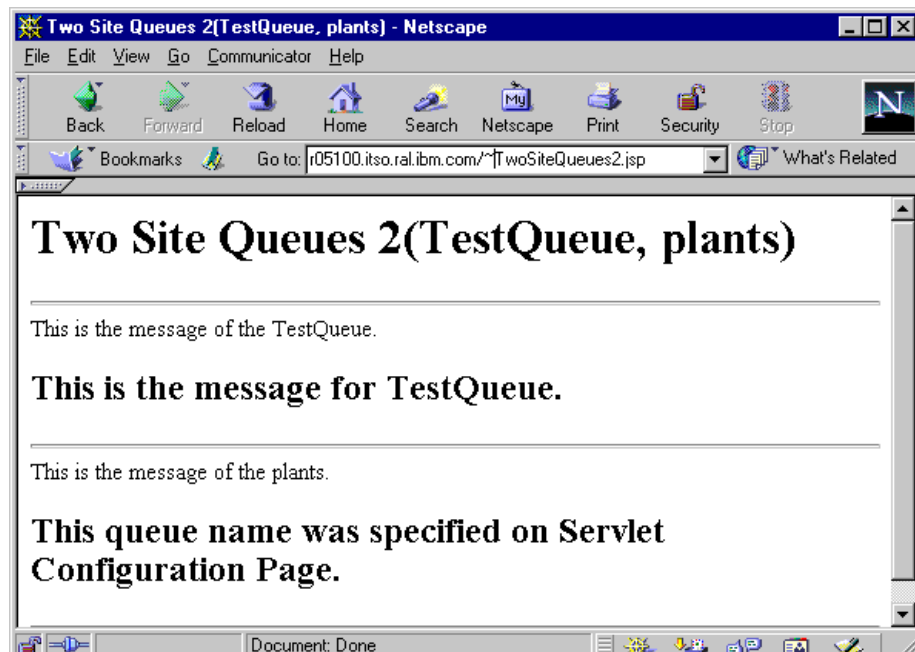


Figure 291. The `queueName` Parameter of the `<SERVLET>` Tag Is Ignored

The servlet uses the parameter values from where it is loaded. Therefore, we recommend that you specify the same parameters for both the `<SERVLET>` tag and the Servlet Configuration page.

5.3.2 Web Site Messaging

SendMessage, CheckMessage and GetMessage servlets in the package com.ibm.servlet.servlets.personalization.util enable Web site users to exchange messages on your Web server.

- The SendMessage servlet checks the users on the Web site using the session object and creates a Java script to send the message back to the SendMessage servlet (Figure 292 on page 295). The SendMessage servlet stores the message in the Message property of the session object (the instance of IBMSessionData).

```
<SCRIPT LANGUAGE="JavaScript" >

function sendMessage() {
SM=window.open('', 'SendMessage', 'scrollbars=no,width=400,height=200');
SM.document.write("<FORM METHOD='POST' ACTION='/servlet/SendMessage'");
SM.document.write("<P>Recipient: <SELECT NAME='userID'>");
SM.document.write("<OPTION SELECTED> Choose a person ");
SM.document.write("<OPTION>wakako")
SM.document.write("<OPTION>fox")
SM.document.write("</SELECT>" );
SM.document.write("<P>Message to send: <INPUT TYPE='option'
NAME='messageText'>");
SM.document.write("<P><INPUT TYPE='submit' VALUE='Submit Message'
onClick='document.forms[0].submit()' >");
SM.document.write("</FORM>");
}
}
```

Figure 292. Java Script Created by SendMessage Servlet

- The CheckMessage servlet checks whether the message is stored in the user's session object. If there is a message, the CheckMessage servlet renders the MessageWaiting image and it adds a hot link to the GetMessage servlet:

```
<a href="/servlet/GetMessage">
```

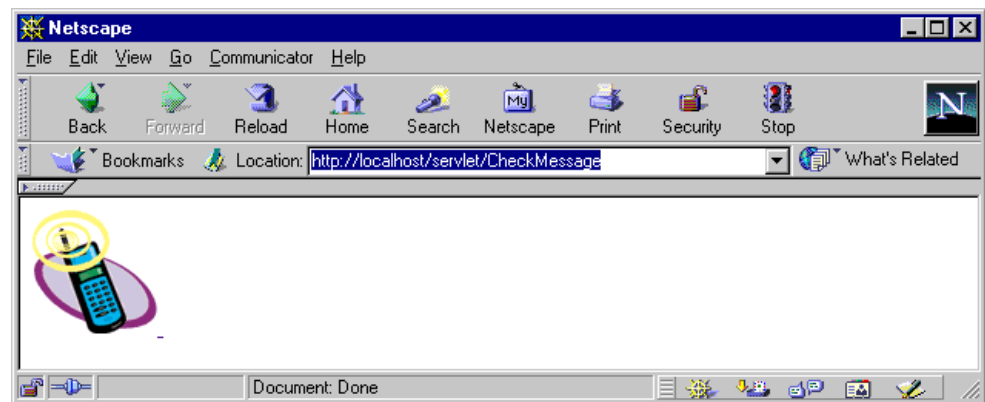


Figure 293. If There Is a Message

If there isn't any message it renders the NoMessageWaiting image:

```

```

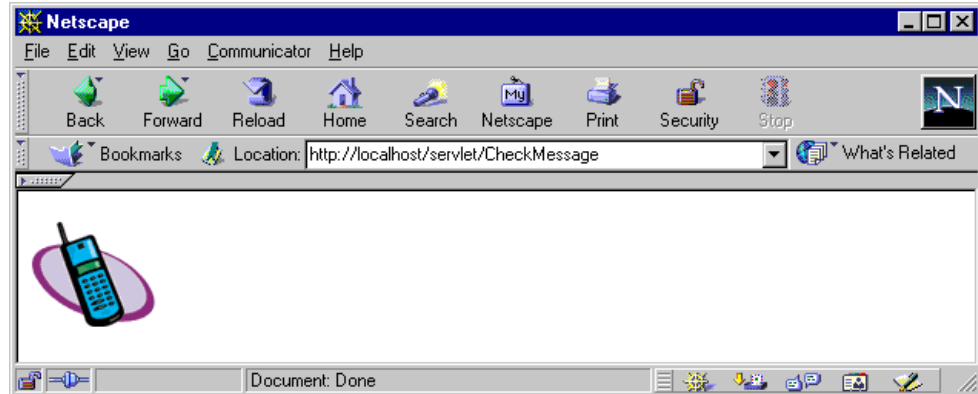


Figure 294. If There Is No Message

- The GetMessage servlet is called from the MessageWaiting image link. It checks and presents the Message property of the user's session object.

5.3.2.1 Sample Model of Message Exchanging

We show a sample scenario exchanging messages below:

1. Only the users registered to user profile can open the interface to exchange messages.
2. Two users are logged on to this Web server to use this application.
 - The MessageLogin.html (Figure 295 on page 297) passes the user name from the form to the MessageLogin servlet.
 - The MessageLogin servlet (Figure 296 on page 298 and Figure 297 on page 299) creates the session object by storing the user name and it retrieves the user profile by that name. If the user has the user profile, this servlet presents the ExchangeMessage.jsp. If not, the session object is invalidated and the user can't log in.
 - The MessageLogin.jsp (Figure 298 on page 300) calls SendMessage and the CheckMessage servlet with the <SERVLET> tag. The user can send or check the message on this screen.

```
<html> <head>
<title>Message Login</title>
</head>
<BODY>
<h1>Message Login</h1><p><br>
<form action = "/servlet/MessageLogin" method = "Get">

Input <B>User Name</B> and Click login button.<P>
<input type = "text" name = "userName"><P>
<tr><td><INPUT TYPE="submit" VALUE="login" >

</form>
<hr>
<!-- hhmts start -->
Last modified: Tue Jun 01 10:44:27 1999
<!-- hhmts end -->
</body> </html>
```

Figure 295. MessageLogin.html

```

import com.ibm.servlet.personalization.sessiontracking.*;
//import com.ibm.servlet.personalization.userprofile.*;
import UserProfile3;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MessageLogin extends HttpServlet {
    public void service(javax.servlet.http.HttpServletRequest
req,javax.servlet.http.HttpServletResponse res) throws ServletException,
IOException {
        String button = null;
        String userName = null;
        String message = null;

        //    UserProfile up = new UserProfile();
        UserProfile3 up = new UserProfile3();
        Hashtable userInfo = new Hashtable();

        //    button = req.getParameterValues("button")[0];
        userName = req.getParameterValues("userName")[0];

        //-----IBMsession-----
        // Step 1: Get the Session object casting to IBMSessionData
        IBMSessionData isession =(IBMSessionData) req.getSession(true);
        // Step 2: Get the session data value
        Integer ival = (Integer) isession.getValue ("MessageLogin.counter");
        if (ival == null){
            ival = new Integer (1);
            isession.setUserName(userName);
        }
        else{
            ival = new Integer (ival.intValue () + 1);
        }
        isession.putValue ("MessageLogin.counter", ival);
        //-----IBMsession-----

```

Figure 296. MessageLogin.java (1/2)

```

        if((up
= (UserProfile3)UserProfile3.retrieveUserProfileByUserName(userName)) ==
null){
message = "No Data Exists for <B> "+userName+" </B><BR> Login denied<BR>";
isession.invalidate();
        }else{
message = "Search Finished";
((com.sun.server.http.HttpServiceResponse)
res).callPage("/ExchangeMessage.jsp", req);
return;
        }

//      up.updateUserProfile(userInfo);

res.setContentType("text/html");
res.setHeader("Pragma", "No-cache");
res.setHeader("Cache-Control", "no-cache");
res.setDateHeader("Expires", 0);
PrintWriter pw = res.getWriter();

pw.println("<HTML><HEAD>");
pw.println("<TITLE>Login Denied</TITLE></HEAD>");
pw.println("<BODY>");
pw.println("<H1>Login Denied</H1>");
pw.println("<HR WIDTH=\"100%\">");
pw.println("<BR>" + message + "<BR>");
pw.println("<B>You have hit this page " + ival + " times" +
"</B><br>");
pw.println("<FORM ACTION = \"/MessageLogin.html\" METHOD = \"GET\">");
pw.println("<P><INPUT TYPE = \"submit\" VALUE = \"Back\"></FORM>");
pw.println("</BODY></HTML>");
pw.flush();
pw.close();
    }
}

```

Figure 297. MessageLogin.java (2/2)

```

<html> <head>
<title>Exchange Message</title>
</head>
<BODY>

<h1>Exchange Message</h1>
<hr>
<h2>Send Message</h2>
<SERVLET NAME="SendMessage">
</SERVLET>
<B> Click The Button and Send User Message!</B>
<hr>
<h2>Check Message</h2>
<SERVLET NAME="CheckMessage">
</SERVLET>
<P><B> Click The Phone and Get Your Message!</B>
<hr>
</body> </html>

```

Figure 298. ExchangeMessage.jsp

To use this sample:

1. Copy MessageLogin.html and ExchangeMessage.jsp into your HTTP document root. For example, the default document root for IBM HTTP Server V1.3.3 on Windows NT is \Program Files\ IBM HTTP Server\htdocs.
2. Copy MessageLogin.java into <ASRoot>/servlets. Compile MessageLogin.java and create the MessageLogin.class file:

```

C:\WEBSPH-1\APPSER-1\servlets>javac MessageLogin.java

C:\WEBSPH-1\APPSER-1\servlets>

```

3. In this scenario, the users wakako and fox are already registered in the user profile. To make a user profile, please refer to 5.2.3, "UserProfile Sample" on page 256.
4. Open the URL <http://<hostname>/MessageLogin.html>. The following window appears:



Figure 299. Login Interface for Exchanging Messages

5. The user wakako was logged in from this window.
6. The user fox was logged in to the server from another client machine.
7. The user wakako sees the following window (Figure 300 on page 302). The noMessageWaiting image is rendered below the Check Message string, since this user hasn't received the message yet.



Figure 300. If the User Name Exists in the User Profile, This Windows Will Appear

8. Click the **Send User Message** button to send a message to someone.
9. The following window comes up. Click the **Recipient** pull-down menu and the users with a session object are shown (Figure 301):

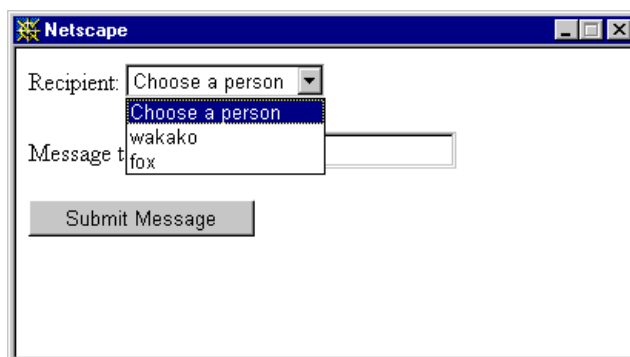


Figure 301. Sending a Message

10. If you open the Active Session Monitor page at that time (click **Server Execution Analysis -> Monitors -> Active Sessions** from the Application Server Manager), the following sessions will be monitored:

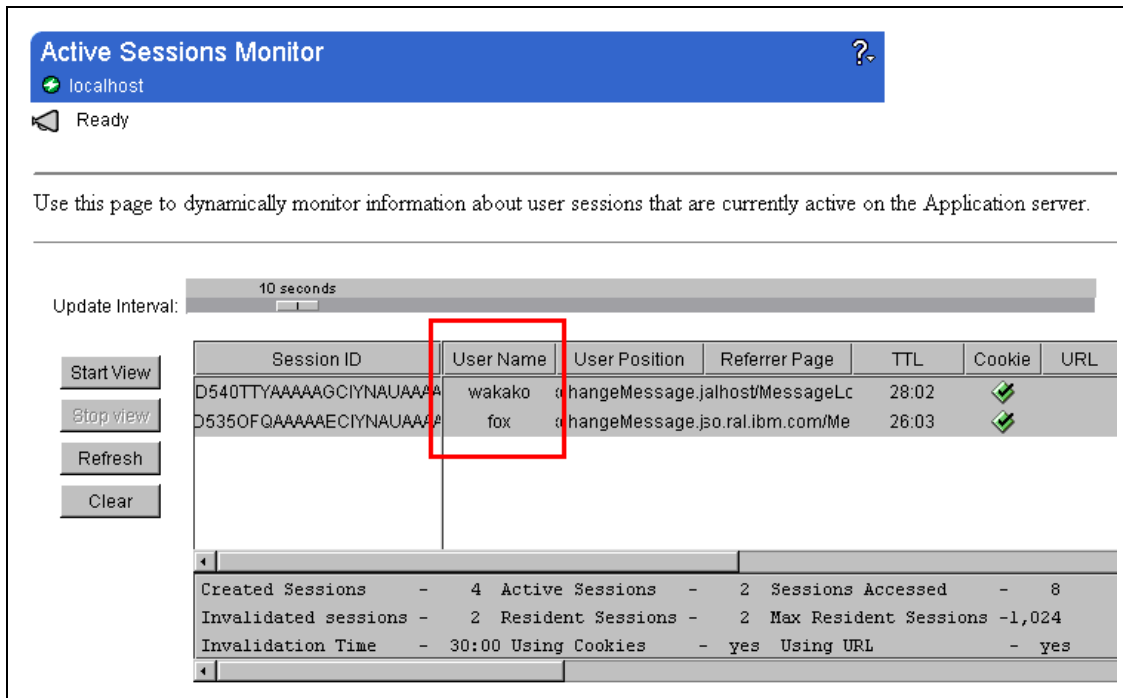


Figure 302. Active Session Monitor

11. The user wakako chooses the user fox, specifies the message and clicks the **Submit Message** button.

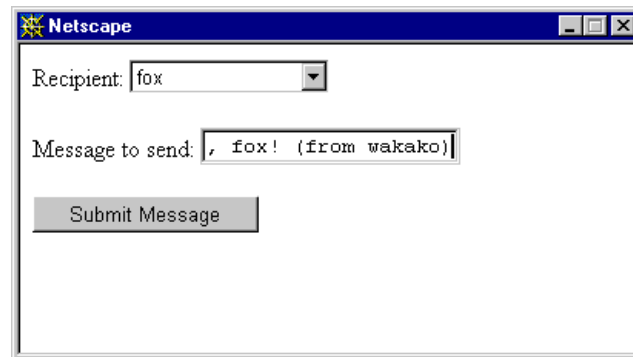


Figure 303. Input the Message and Click the Submit Message Button

12. The following dialog box comes up. Click **OK**.

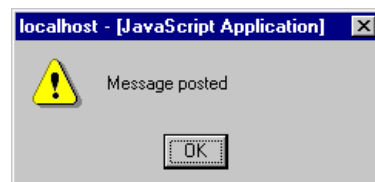


Figure 304. Message Posted

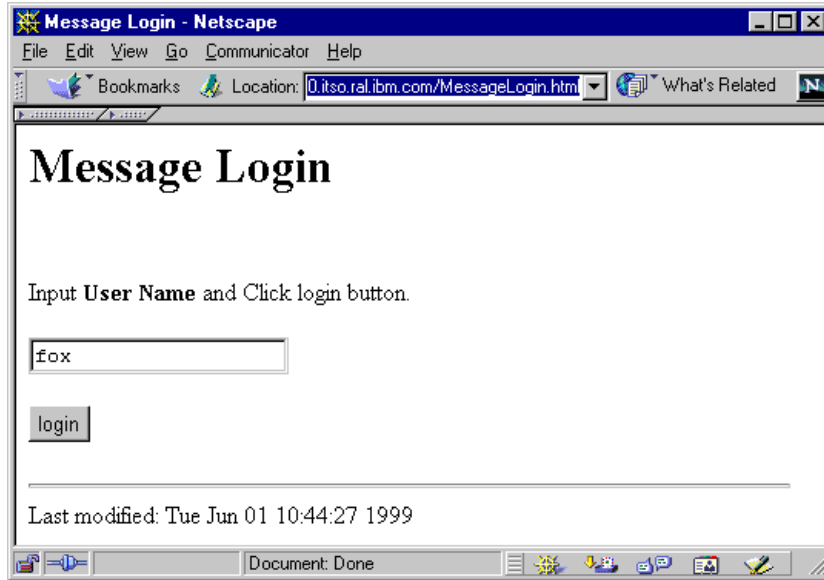


Figure 305. "fox" Login from Another Machine

13.If the user fox logs in after these steps (Figure 305), the following window will appear. The *messageWaiting* image will be rendered with a hot link to the GetMessage servlet (Figure 306).



Figure 306. Fox Has Message

14. Click the image or the string below the image and you can get the following message:

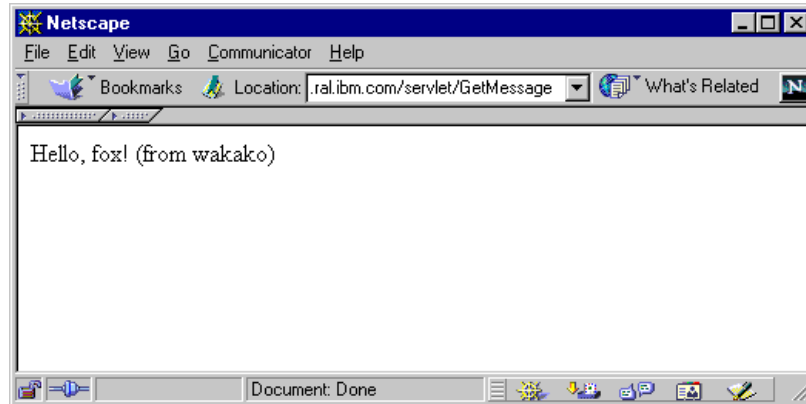


Figure 307. The Message from wakako

15. If you reload this page, you will get the following message (Figure 308), since the GetMessage servlet clears the message after retrieving it:



Figure 308. The Message is Disabled

16. If the user is not registered to the user profile, the following window will appear and the user's session is invalidated from the MessageLogin servlet:

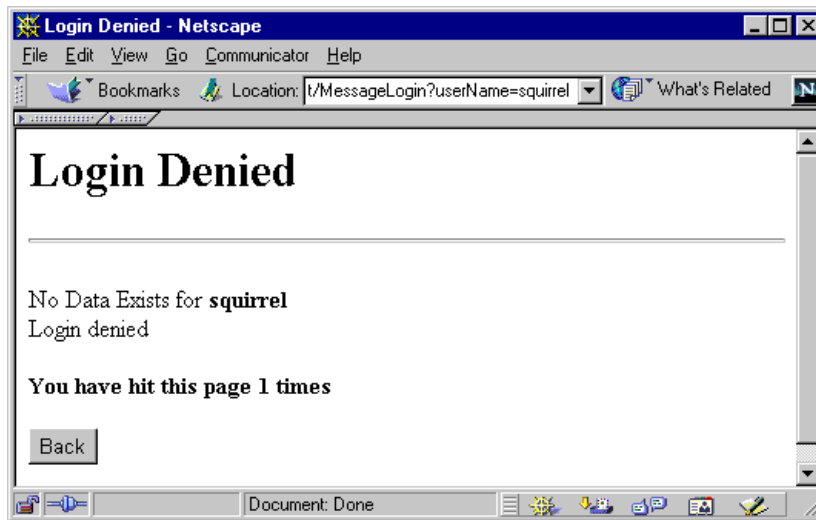


Figure 309. Login Denied

5.4 Connection Pooling

Connecting and disconnecting from a data server causes you to use resources inefficiently. The connection manager lets you connect and disconnect to a data server more efficiently by pooling connections and reusing them. That reduces the overhead of connecting and disconnecting

The servlets use the connection pool as follows:

When a user makes a request over the Web to a servlet, the servlet uses an existing connection from the pool, meaning the user request does not incur the overhead of a data server connection. When the request is satisfied, the servlet returns the connection to the connection manager pool for use by other servlets. The user request, therefore does not incur the overhead of a data server disconnection.

The connection manager also lets you control the number of concurrent connections to a data server product. This is very useful if the data server license agreement limits you to a certain number of concurrent users.

There is a good introduction to connection manager in the WebSphere documentation at [http://\[hostname\]/IBMWebAS/doc/whatis/iccmgr.html](http://[hostname]/IBMWebAS/doc/whatis/iccmgr.html).

5.4.1 Key Terms

Data server refers to many different data sources, including:

- Relational databases such as DB2, Oracle, Informix, and Sybase.
- Other types of products, whose special services for managing and accessing data vary by product.

Connection Manager Connection - We use this term to distinguish from "real" connections to the underlying data server, for example, a database or MQ.

5.4.2 Connection Manager Architecture

The Connection Manager maintains a pool of open data server connections to specific data server products. Each data server can have one or more identical or non-identical pool(s). Multiple data servers can be supported by one running instance of the connection manager.

Figure 310 illustrates the typical interactions between the connection manager and a servlet seeking to use a connection from the connection manager's connection pool. The following list traces the steps that are followed when a connection request is made. The sequence changes if a connection is not available when the servlet requests one.

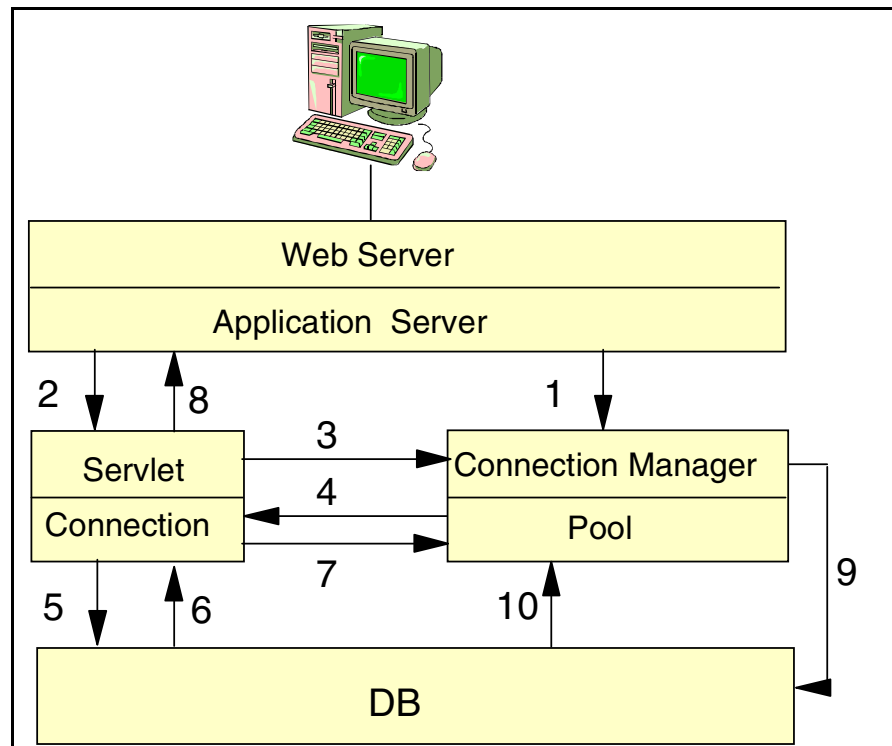


Figure 310. Connection Manager Architecture

1. The connection manager, which runs under the application server, is loaded by the application server when the first servlet tries to communicate with the connection manager. The connection manager stays loaded as long as the application server is running.
2. The application server passes a user request to a servlet.
3. The servlet uses methods from the connection manager to request a connection from the pool.
4. The pool gives the servlet a connection.
5. The servlet uses the connection to talk directly to the data server, using the standard APIs for the specific data server.
6. The data server returns data through the connection to the servlet.

7. When the servlet ends communications with the data server, the servlet returns the connection to the pool for use by another user request.
8. The servlet sends the response back through the application server to the user.

5.4.2.1 Monitoring the Connection Manager

The application server manager provides a monitor for the connection manager called DB Pool Connections. See the application server help menus for a detailed description of the information the monitor provides. You can use that information to see how connection pools are performing and to suggest possible changes in the connection pool parameters. You can monitor a pool after you change the parameters in order to see the change in pool behavior and to assist you in further tuning of the pool. You can see this monitor by clicking **Server Execution Analysis -> Monitors -> DB Pool Connections** (Figure 311):

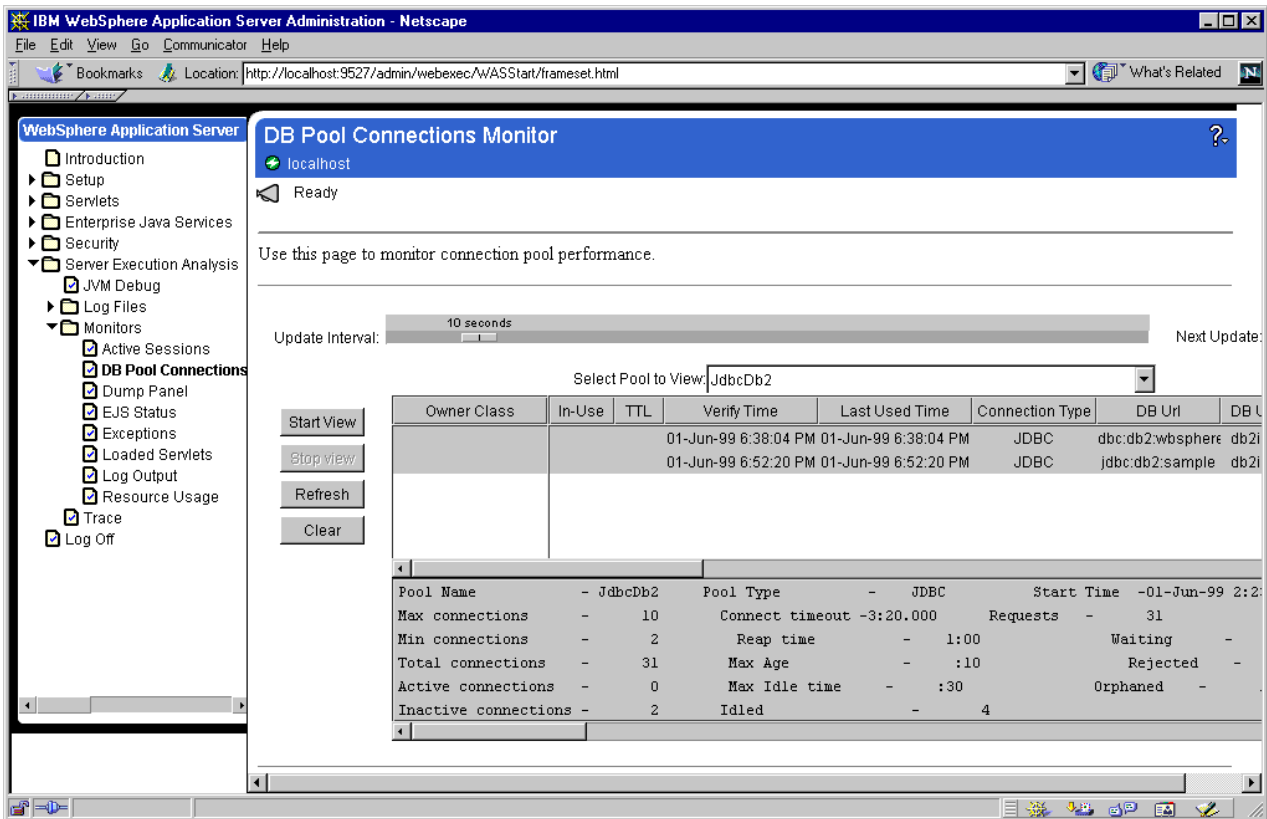


Figure 311. DB Pool Connections

You can select the specific pool to monitor from a selection list (Figure 312).

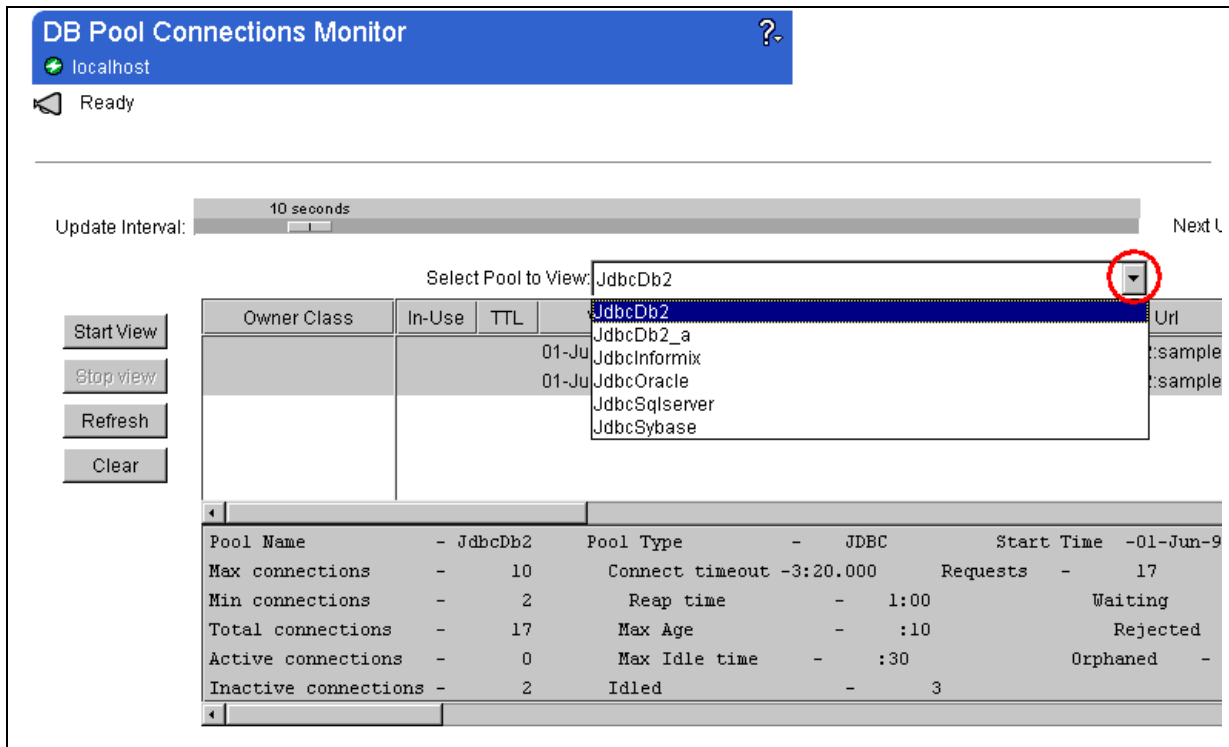


Figure 312. You Can Select the Specific Pool

The resulting monitor display consists of:

- Each connection in the pool
- Summary information about the entire pool

5.4.2.2 Each Connection in the Pool

Information provided by the connection list includes:

- Owner Class

The servlet class that currently owns the connection (or the servlet class that last released the connection, if it is free) as shown in Figure 313.

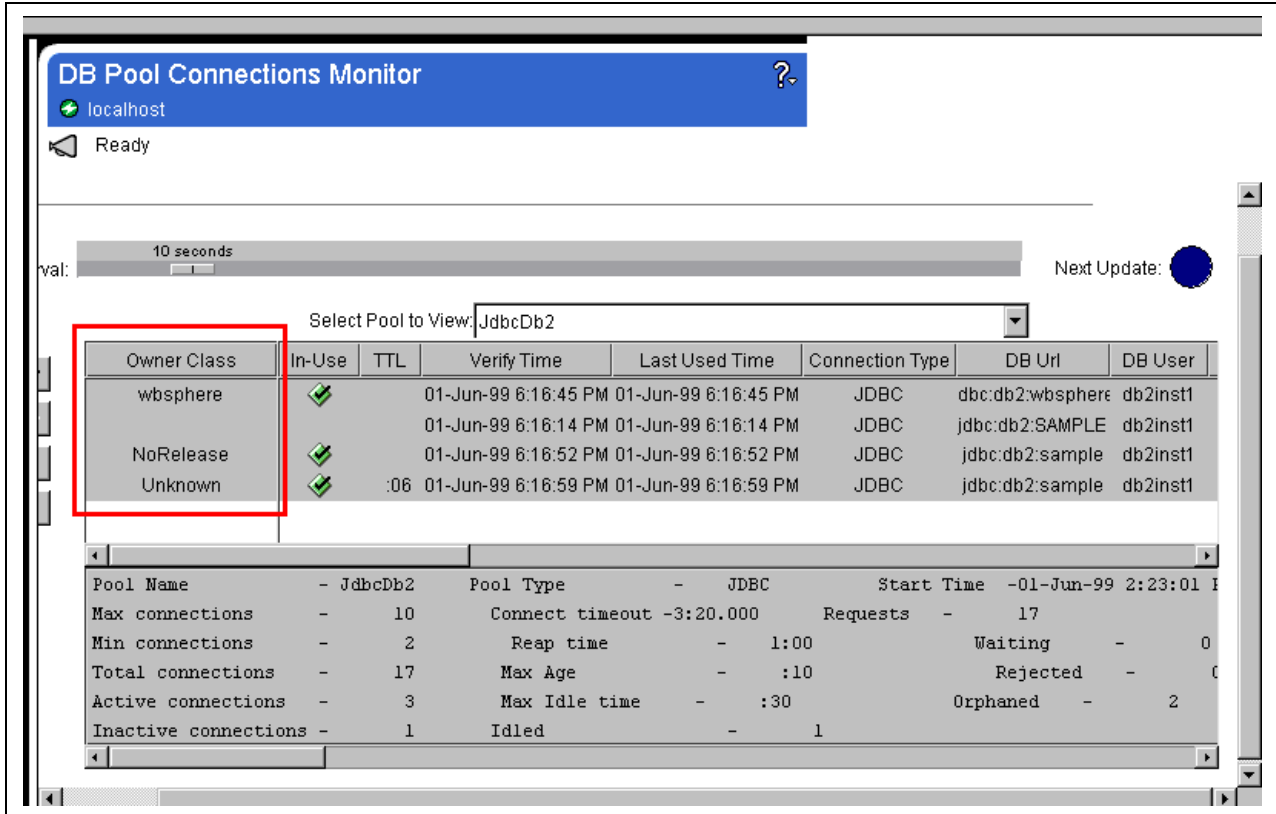


Figure 313. Owner Class Field

You can get a class name that is known to the connection manager by using the `getIBMConnection(IBMConnSpec connSpec, String ownerClass)` method in your servlet. `NoRelease.java` uses this method as you can see below. If you use the `getIBMConnection(IBMConnSpec connSpec)` method, then "Unknown" appears in this field.

A Piece of NoRelease.java

```
cmConn0 =
(IBMJdbcConn)connMgr.getIBMConnection(spec0,"NoRelease");
```

- In-Use

This determines if the connection is in use by a servlet. If the connection is currently free or available for a servlet connection, then this field is blank. Otherwise, the field contains a check mark (as shown in Figure 313).

- Verify Time

The verify time stamp and the last used time stamps for the connection.

- DB URL, DB User, Driver Class

Additional information, some of which may depend on the type of data server. The fields, for example DB URL, are specific to each JDBC data server.

5.4.2.3 Summary Information about the Entire Pool

Information provided by the pool summary includes:

- Pool Name

This is the unique name of a pool, used by the servlet programmer to access the pool and used within the monitor to identify a pool and the statistics associated with it.

- Total Connections

This is a cumulative total of the successful requests for connections made to the pool.

- Requests

This is a cumulative total of both the successful and the unsuccessful requests for connections made to the pool.

- Waiting

This is the number of connection requests currently waiting for a connection from the specified pool. When a servlet's `waitRetry` parameter is set to true, the servlet can wait for a short time for a connection if the pool doesn't have one immediately available. Requests waiting for a connection will fail if the connection is not made available within the time specified by the connection timeout parameter. See the rejected statistic below for the number of requests that actually failed. If the servlets using this pool are important, you may want to increase the maximum connections parameter for the pool to reduce the chances that a connection request will have to wait.

- Rejected

This is the number of connection requests that were refused a connection. It is the cumulative total of the waiting requests that failed to get a connection, plus failed requests from servlets whose `waitRetry` parameter was set to false and thus failed right away when a connection was not available. Generally, you will want *rejected* to be a small number compared to total connections, particularly if important servlets are using the pool. Look at increasing the maximum connections and the connection timeout parameters of the connection pool to keep rejected to a small percentage of total connections.

- Orphaned

This is the number of connections taken away from servlets that have died or otherwise become unresponsive, or taken away from servlets that may have been coded improperly. The connections are taken away from the servlets in the periodic reap process and are returned to the pool so that other servlets can use them. Something is probably wrong if the orphaned statistic is something other than zero, and you should look at making some servlet coding changes. If the value is other than zero, it may mean that connections are not being used and yet the connections are not available to be used by a new request. This can happen if a servlet fails without explicitly releasing connections that it owns, or if the servlet performs normally but the programmer neglects to explicitly release the connection after sending the response back to the user. The servlet should probably be changed to use the `releaseIBMConnection()` method in case of a servlet failure and at the end of a successful response. The orphaned statistic can also be greater than zero if a servlet neglects to periodically verify with the connection manager that it still needs to hold a connection over an extended period of time.

For example, we tested the "NoRelease" servlet that doesn't execute the releaseIBMConnection method. See Table 30 for the results of that test:

Table 30. DB Pool Setting

Parameters	Values	Parameters	Values
Maximum Connections	10	Maximum Age	10
Minimum Connections	2	Maximum Idle Time	30
Connection Timeout	200000	Reap Time	60

After executing this servlet, we found this servlet using CM connection in DB Pool Monitor (Figure 314):

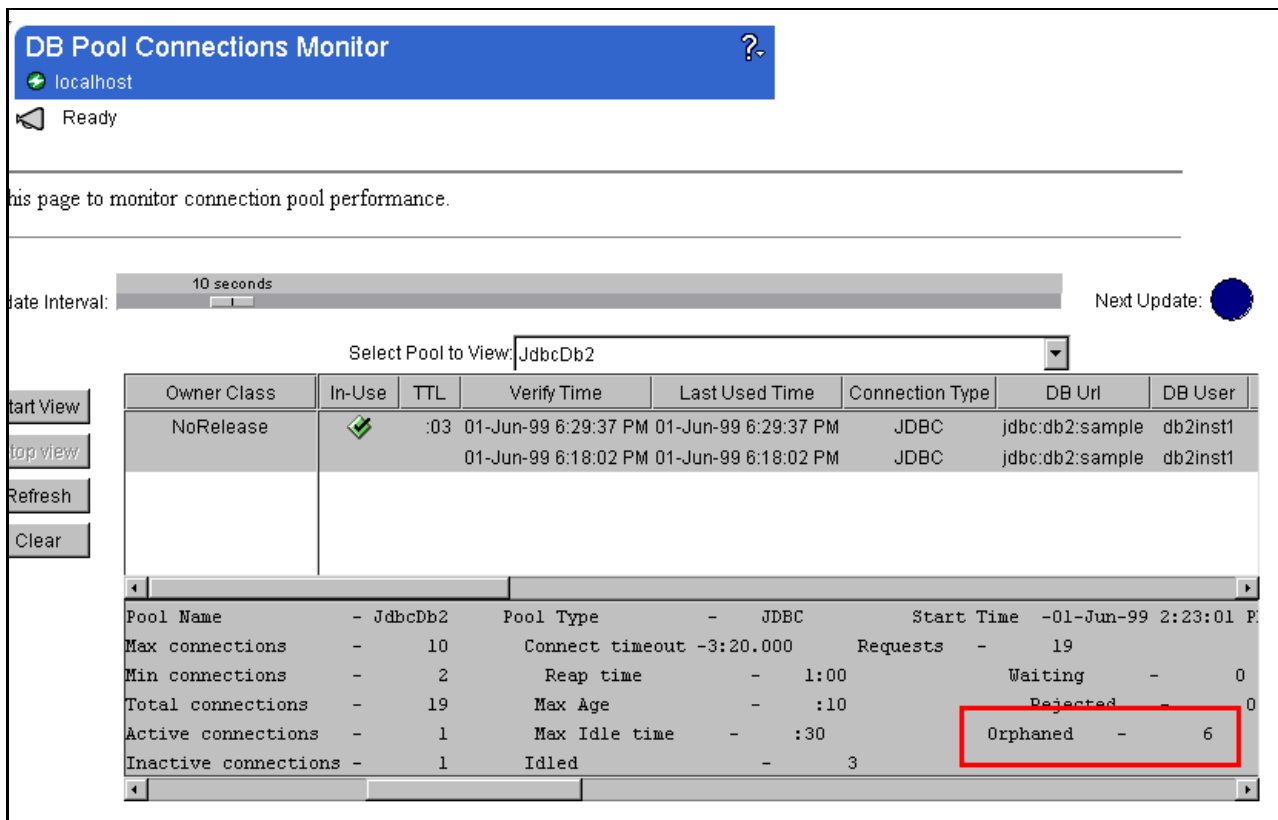


Figure 314. "No Release" Servlet That Doesn't Release the Connection

At that time, we found that the orphaned statistic was not zero (Figure 314).

- Idled

This is the cumulative number of connections that the reap process has removed from the pool (and disconnected from the data server). The connections may be removed after they remain idle (unassigned to any servlets) beyond a certain time. If the Inactive statistic is high compared to total connections, it may mean that there is a lot of connect/disconnect overhead compared to the number of requests actually serviced. This in turn might be due to excessive fluctuations in the number of user requests. Connections are created to satisfy a temporary peak, and then discarded

during a temporary lull. You might want to make the pool less sensitive to such fluctuations by increasing the reap time or maximum idle time pool parameters.

5.4.2.4 Managing Connections

Use the Connection Management page to define connection pools corresponding to data servers, such as DB2. The application server connection manager lets user servlets borrow data server connections from these pools, helping you control and reduce the resources used by your Web-based applications.

Viewing the Pool Types Supported by the Application Server

Click the **Pool Types** tab to view the types of data servers supported by the connection manager. Currently, Java Database Connectivity (JDBC)-compliant databases are supported.

Adding Connection Pools

To add connection pools:

1. View the **Setup -> Connection Management** page. (see Figure 315.)
2. Click the **Pool List** tab.

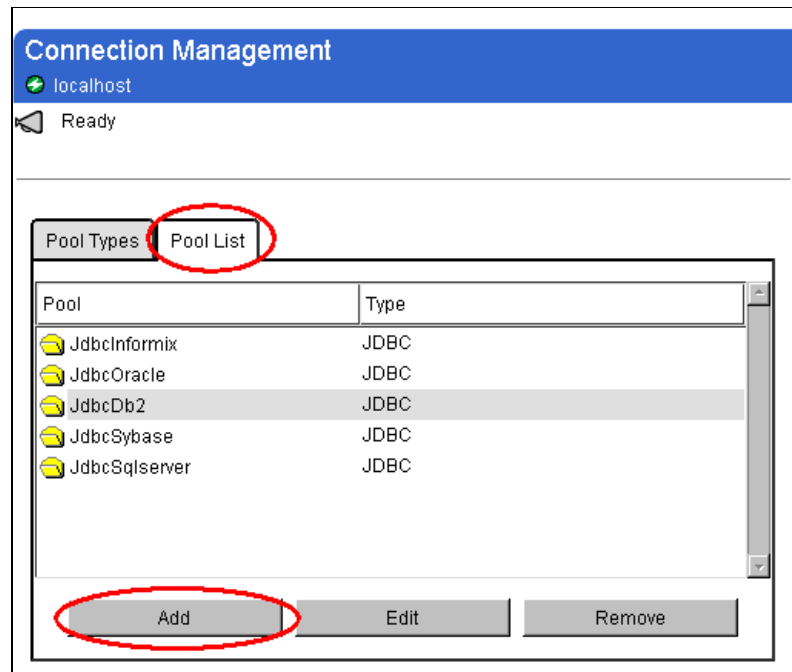


Figure 315. Pool List

3. Click the **Add** button.
4. In the Pool Name field, specify the pool to create (Figure 316).

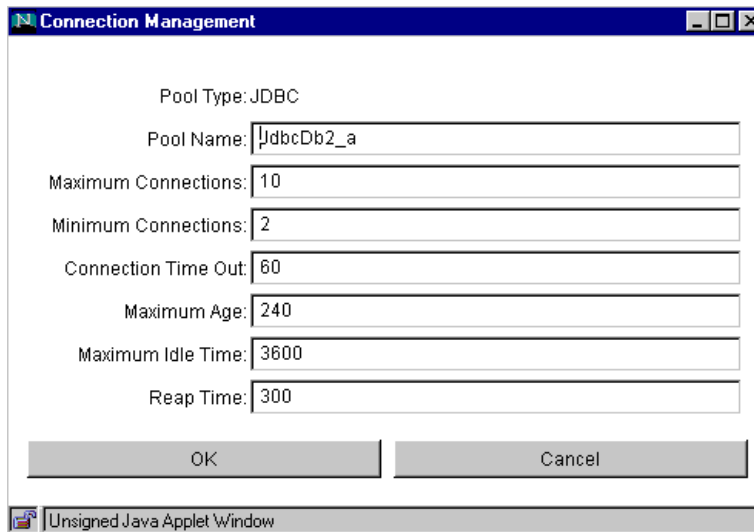


Figure 316. Pool Configuration

5. In the other fields, enter your preferred configuration settings.
6. Click the **OK** button.

Note: Tell your servlet programmers the Pool Name so they can use it in the servlets that access the connection pool.

5.4.2.5 Each of the Fields in the Pool

You can see this field information at

<http://<hostname>:9527/admin/webexec/WASHelp/hfcm.htm>.

- Pool Type

The type of data server used by this pool of connections. JDBC indicates a JDBC-compliant database. A data server is a product that helps you manage and access data. Usually, it is a relational database such as DB2, Oracle, Informix, or Sybase.

- Pool Name

The unique name for this pool of connections. Servlet programmers need to know this name for their servlets to use this connection pool.

- Maximum Connections

Maximum number of connections that can be in the pool. Consider setting it to the maximum number of users permitted by your data server product license agreement.

- Minimum Connections

The minimum number of connections that can remain in the pool as a result of the reap process. The connection manager periodically removes connections that become idle or orphaned. Use this setting to keep from removing too many connections, erasing resource usage performance gains.

- Connection Time Out

The length of time (in milliseconds) the connection manager will wait for a connection to become free when all connections in the pool are currently in

use and the number of connections has reached the maximum (meaning no new connections can be created to fulfill the need).

- A value of 0 allows the connection manager to wait forever.
- A value of -1 disables the wait (an exception is immediately thrown if a connection is not available).
- A value of 1000 to 2000 (1 to 2 seconds) is suggested.

Servlet programmers must know the value of the connection timeout to effectively set the `waitRetry` parameter in their servlets.

- **Maximum Age**

The maximum number of seconds a connection can be idle before the reap process releases the connection from the servlet that owns it.

- A value of -1 disables this function; the reap process will not release any connections from idle servlets.
- A value of 900 to 1800 (15 to 30 minutes) is suggested.

- **Maximum Idle Time**

The maximum number of seconds an unassigned connection can remain in the pool.

- A value of -1 disables this function; the reap process will not remove any connection from the pool and disconnect it from the data server.
- A value of 900 to 1800 (15 to 30 minutes) is suggested.

- **Reap Time**

The interval (in seconds) at which the connection manager performs the reap process.

- A value of -1 disables this function; the reap will not be performed, regardless of the values specified in the Maximum Age or Maximum Idle Time parameters.
- A value of 1800 to 3600 (30 to 60 minutes) is suggested.

5.4.2.6 How to Manage CM Connections When the DB Server Is Down

When CM connections lose DB connections for some reason (for example, a DB server goes down or there are network problems), the connection objects still remain in the pools.

If we execute an SQL query with this connection, an SQL exception would be thrown even after the DB connections were recovered. For example, the SQL exception might look like the following:

```
get connection, process statement: [IBM] [CLI Driver] [DB2/NT] SQL1224N A data agent could not be started to service a request, or was terminated as a result of a database system shutdown or a force command.  SQLSTATE=55032
```

When the DB connection was terminated from the DB2 command line processor, the DB Pool Connection seemed OK since the DB connections still existed (see Figure 317 on page 316).

```

C:\WebSphere\AppServer\classes>cd \

C:\>db2 list applications

Auth Id  Application      Appl.      Application Id      DB      # of
        Name          Handle                               Name    Agents
-----
DB2INST1 java.exe      142       *LOCAL.DB2.990601224056  SAMPLE  1
DB2INST1 java.exe      141       *LOCAL.DB2.990601223702  WBSPHERE 1

C:\>db2 force application all
DB20000I The FORCE APPLICATION command completed successfully.
DB21024I This command is asynchronous and may not be effective immediately.

C:\>db2 list applications
SQL1611W No data was returned by Database System Monitor.

```

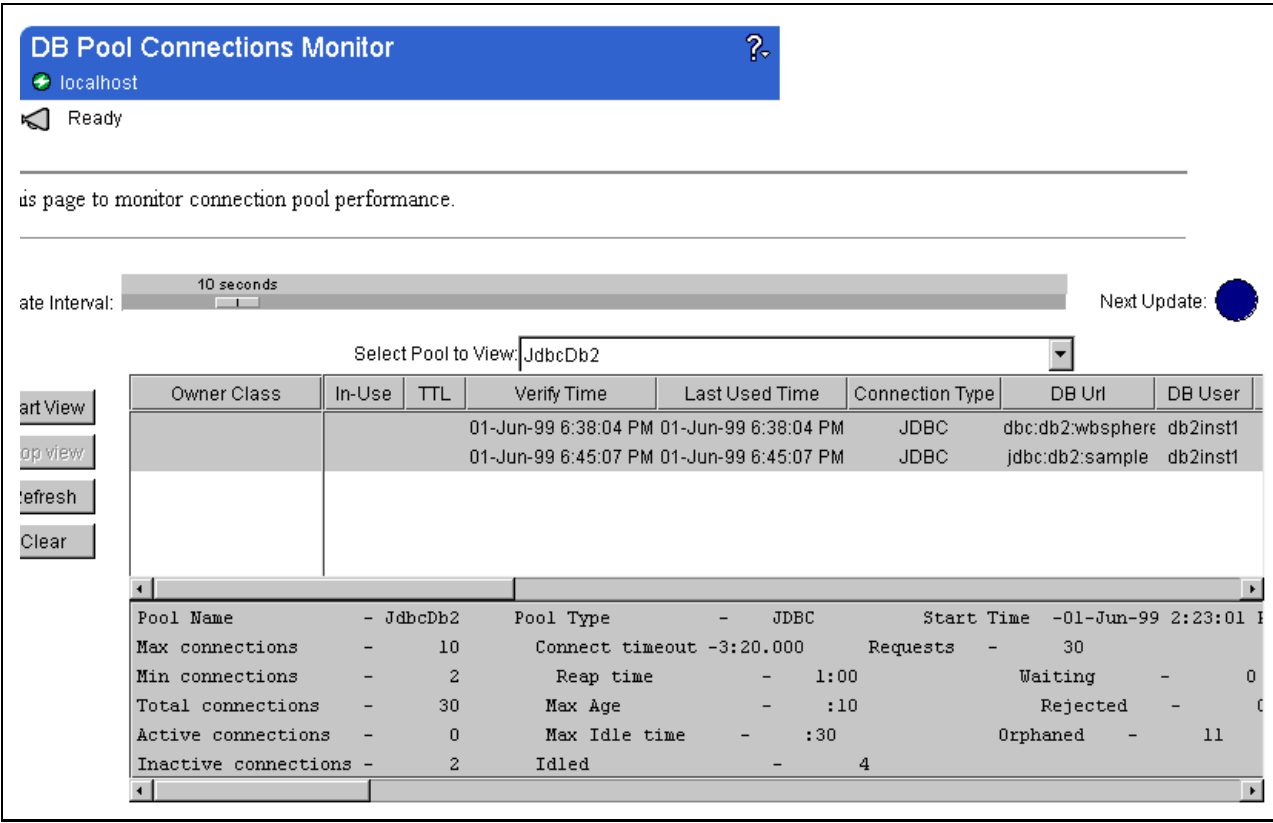


Figure 317. The Connections in the Pool

The servlet can't connect to the database using this connection (Figure 318 on page 317):

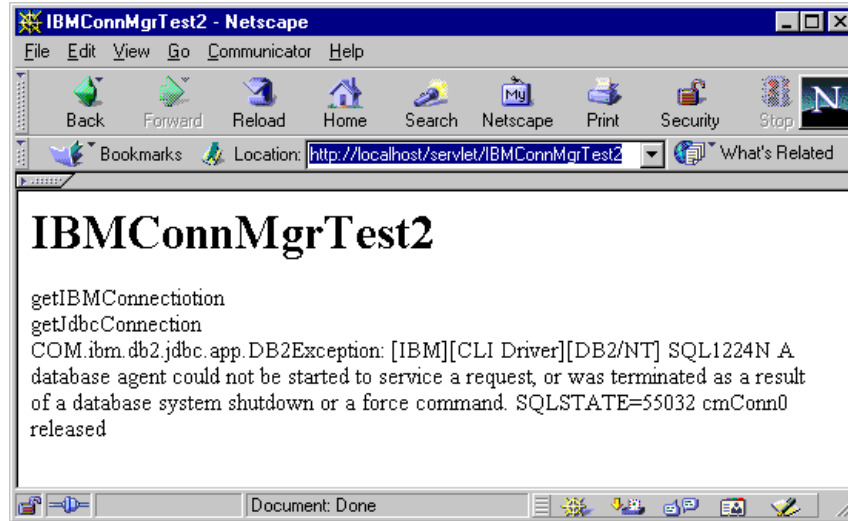


Figure 318. SQL Exception

We can solve this no DB connection problem with the `removeIBMConnection` method of the `com.ibm.servlet.connmgr.IBMConnection` class. This method removes the connection handle from the pool. We should design all servlets to implement this method when they catch an SQL exception.

```

rs.close();
stmt.close();
}catch(Exception e){
    pw.println("Exception!<BR>");
    e.printStackTrace(pw);
//*****
// remove IBMConnection
//*****
    try{
        cmConn.removeIBMConnection();
        pw.println("removeIBMConnection...<BR>");
    }catch(Exception e2){
        pw.println("Exception at removeIBMConnection<BR>");
        pw.println(e.toString()+"<BR>");

    }//end of catch e2
    pw.flush();
} //end of catch e

```

Figure 319. The `removeIBMConnection` Method

The one thing we must be careful about is that this method removes only one connection handle at a time. If you want to remove all of the connection objects, you must implement proper program procedures.

Note: You can use the `removeIBMConnection` method with the WebSphere Application Server V2.X API or later. WebSphere V1.X doesn't implement this method.

5.4.2.7 Reap Process

The connection manager removes idle connections from the pool because they waste resources. To decide which connections are idle the connection manager checks the connection flags and time stamps in a periodic reap of the connection pool. There are two kinds of reap:

1. Reap Connections from the Servlet

The connection manager looks at the last-used time stamp of the in-use connections. If the time between the last-used and current time exceeds the maximum age configuration parameter, the connection is assumed to be an orphaned connection, meaning the owning servlet has died or is otherwise unresponsive. The orphaned connection is returned to the pool for use by another servlet, its in-use flag is set to false and its verify and last used time stamps are set to the current time.

2. Reap Idle Connections from the Pool

The connection manager examines connections not in use by any servlet, that is, those connections whose in-use flags are false. If the time between the last-used and current time exceeds the maximum idle time configuration parameter, the connection is assumed idle. Idle connections are removed from the pool, down to the lower limit specified by the Minimum Connections configuration parameter.

We tested this function in the following setting:

- Change the *JdbcDb2* Pool setting as follows, so that we can evaluate the reap process and get sensitive status changes:

Table 31. DB Pool Setting

Parameters	Values	Parameters	Values
Maximum Connections	10	Maximum Age	10
Minimum Connections	2	Maximum Idle Time	30
Connection Time Out	200000	Reap Time	60

- Executed servlets named.
 - NoRelease - This servlet doesn't execute the `releaseIBMConnection()` method. This causes the problem that:
 - Some connections are not being used, and are not available.
 - The orphaned statistic is not 0. Check the DB monitor.
 - For the orphaned statistic we already showed the result in 5.4.2.3, "Summary Information about the Entire Pool" on page 311.
 - NoRelease2 - This servlet doesn't execute the `releaseIBMConnection()` method. The class name of this servlet appeared as an "Unknown" class because it doesn't tell the servlet name to the connection manager.
 - Websphere - This servlet doesn't execute the `releaseIBMConnection()` method. This opens the connection to the *wbsphere* database, while two of the servlets above connect to the *sample* database.
- Checked DB Pool Monitor.

At first we could see this status on the DB Pool Connections Monitor. As you can see, we can connect to multiple databases in one pool:

Owner Class	In-Use	TTL	Verify Time	Last Used Time	Connection Type	DB Url
wbsphere	✓	:05	05-May-99 4:19:44 PM	05-May-99 4:19:44 PM	JDBC	jdbc:db2:wbsphere
NoRelease	✓		05-May-99 4:19:37 PM	05-May-99 4:19:37 PM	JDBC	jdbc:db2:sample
Unknown	✓	:01	05-May-99 4:19:40 PM	05-May-99 4:19:40 PM	JDBC	jdbc:db2:sample

Figure 320. Servlets Owing Connections

You can get the latest information by clicking the **Refresh** button (Figure 321).

DB Pool Connections Monitor

localhost

Ready

Update Interval: 10 seconds

Select Pool to View: JdbcDb2

Owner Class	In-Use	TTL	Verify Time	Last Used Time	Connection Type	DB Url
			01-Jun-99 6:38:04 PM	01-Jun-99 6:38:04 PM	JDBC	dbc:db2:wbspher
			01-Jun-99 6:52:20 PM	01-Jun-99 6:52:20 PM	JDBC	jdbc:db2:sample

Buttons: Start View, Stop view, Refresh, Clear

Pool Name	JdbcDb2	Pool Type	JDBC	Start Time	-01-Jun-9
Max connections	10	Connect timeout	-3:20.000	Requests	31
Min connections	2	Reap time	1:00	Waiting	
Total connections	31	Max Age	:10	Rejected	
Active connections	0	Max Idle time	:30	Orphaned	
Inactive connections	2	Idled	4		

Figure 321. DB Monitor Refresh Button

After a few minutes, we could see the connection manager reaped CM connections from the pool (Figure 322):

Owner Class	In-Use	TTL	Verify Time	Last Used Time	Connection Type	DB Url
			05-May-99 4:20:36 PM	05-May-99 4:20:36 PM	JDBC	jdbc:db2:wbsphere
			05-May-99 4:20:36 PM	05-May-99 4:20:36 PM	JDBC	jdbc:db2:sample
			05-May-99 4:20:36 PM	05-May-99 4:20:36 PM	JDBC	jdbc:db2:sample

Figure 322. Connection Manager Reaped Connections from Servlets

The minimum connections parameter is two, so two connections remain in the pool. Take note of the fact that the connection to the *wbsphere* database was reaped although it was only connected to this database. That's why we don't

recommend connecting to multiple databases from one pool. If you want to connect to multiple databases, it is easier to manage connections to each database in separate pools.

Owner Class	In-Use	TTL	Verify Time	Last Used Time	Connection Type	DB Url
			05-May-99 4:20:36 PM	05-May-99 4:20:36 PM	JDBC	jdbc:db2:sample
			05-May-99 4:20:36 PM	05-May-99 4:20:36 PM	JDBC	jdbc:db2:sample

Figure 323. Connection Manager Reaped Idle Connections from the Pool

You can create multiple pools for the same database. The pools can be configured to provide different service levels, so that an important servlet using a "high service-level" pool is more responsive than a less critical servlet (Figure 324).

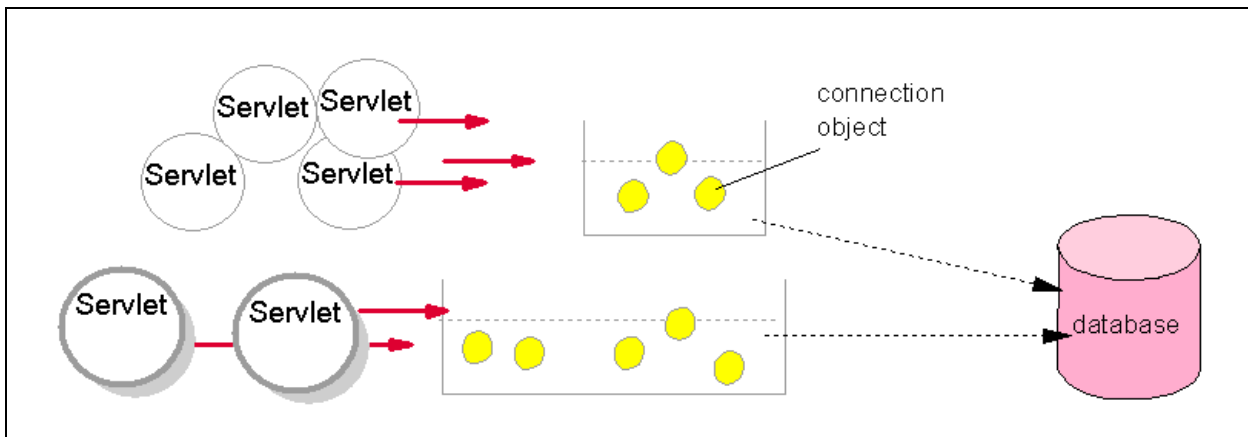


Figure 324. Multiple Connection Manager Pools on the Same Database

5.4.3 Creating Connection Manager Applications

Edit the properties files for the XtremeXML and IBMConnMgrTest samples using the DB2 user ID and password information.

- Copy the file <server root>\samples\login.properties to <server root>\servlets. Edit the file and change the database owner and login parameters. Figure 325 shows an extract from the file with the values change shown in bold. Change DB2owner to the DB2 user ID that you used to create the sample database in step 1 in 5.4.2.7, "Reap Process" on page 318. If you want to access DB2 with a user ID other than the user ID and password that WebSphere uses, change the dbUserid and dbPassword values appropriately; otherwise leave them as *null*. On Windows NT the WebSphere service does not run under any user ID by default, so you must change these values. If you have changed the name of the sample database that you created in step 1 in 5.4.2.7, "Reap Process" on page 318 to something other than *sample*, then change the dbName parameter to that value.

```

XtremAdvXml.dbOwner=DB2owner
XtremAdvXml.dbUserid=YourDB2Userid
XtremAdvXml.dbPassword=YourDB2Password
...
JDBCServlet.dbOwner=DB2owner
JDBCServlet.dbUserid=YourDB2Userid
JDBCServlet.dbPassword=YourDB2Password
JDBCServlet.dbName=sample

```

Figure 325. Lines to Change in <server root>\servlets\login.properties

- Edit the file <server root>\servlets\IBMConnMgrTestStrings.properties and change the values shown in Figure 326:

```

# UserProfileConfig
upc.db=SAMPLE
upc.owner=DBOwner
upc.userid=DBUser
upc.password=DBPassword

# HTML text
# LOCALIZE THIS
html.title=Test for IBMConnMgrTest Servlet
html.nobodyfound=Nobody named Parker
html.greeting=G'day
# END OF MATERIAL TO LOCALIZE

```

Figure 326. Changes in <server home>/servlets/IBMConnMgrTestStrings.properties

Change the upc.db parameter to the name of the sample database (probably sample) and the upc.owner parameter to the owner of the sample database. If you want to log in to DB2 with a user ID and password that is different from the user ID and password that WebSphere runs under, change the upc.userid and upc.password settings appropriately. Otherwise, change them to the word null. On Windows NT the WebSphere service does not run under a user ID and password by default so you must change these values.

If you want to localize the servlet for your country change the html.* parameters to something in your local language. The title heading html.title is used on the servlet. When no results are returned from the database, the message html.nobodyfound, is displayed. To greet someone, html.greeting is used.

IBM Connection Manager Test Servlet

This servlet, while not terribly exciting visually, demonstrates the IBM connection manager performing connection pooling. To run the servlet:

- Click the **Database Servlet** (Figure 327) from the Samples page. Another browser window will come up.



Figure 327. Sample Page ([http:// \[hostname\] /IBMWebAS/samples/index.html](http://[hostname]/IBMWebAS/samples/index.html))

- Click **Run IBMConnMgrTest** (Figure 328). Yet another browser window should come up with a greeting for JOHN Parker(). The form of the greeting derives from the text you entered in the database setup on page 320 for the `html.greeting` parameter.

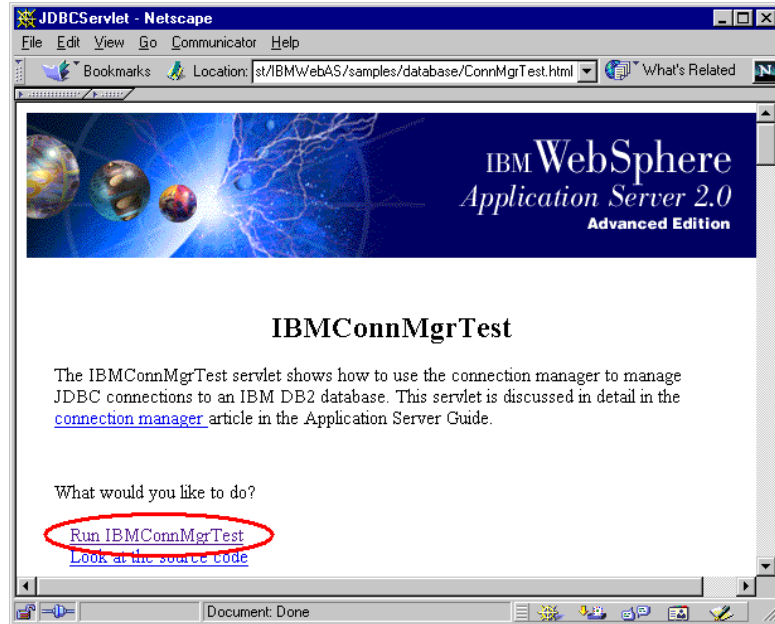


Figure 328. Click Run IBMConnMgrTest

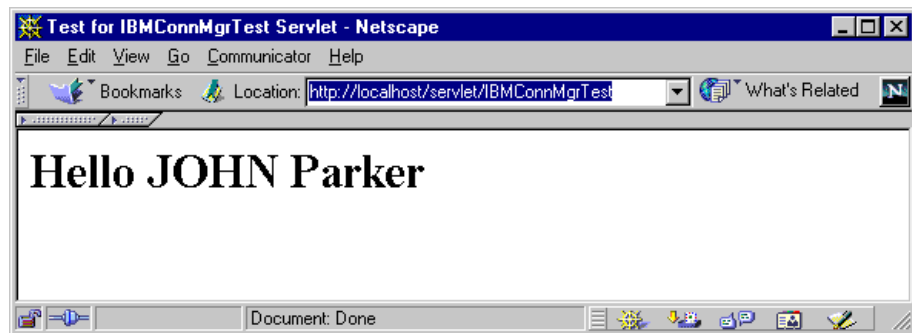


Figure 329. The Result of IBMConnMgrTest

- If something goes wrong you should get the error message that you entered for the `html.nobodyfound` parameter in the database setup on page 320.

5.4.3.1 How Servlets Use the Connection Manager

All servlets using the connection manager will follow these steps; `RmvCon2.java` is used to following these steps.

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import java.util.*;
import java.text.*;
import com.sun.server.util.*;
import com.ibm.servlet.connmgr.*; //for connmgr

/**
 * This is an example of a IBMConnection Manager
 * Servlet.
 */

public class RmvCon2 extends HttpServlet
{
    // for IBMConnMgr
    static IBMConnMgr connMgr = null;
    static IBMConnSpec spec0 = null;
    static String poolName = "JdbcDb2"; // from Webmaster

    // ResourceBundle class constant
    public final String BASENAME = "samples";

    // con is initialized at startup of servlet
    static String url0=null;
    static String Debug = null;

    static String jdbcDriver = "COM.ibm.db2.jdbc.app.DB2Driver";

    static{
        try{
            // register the driver with DriverManager
            Class.forName(jdbcDriver);
        }catch (ClassNotFoundException e){
            e.printStackTrace();
        }
    }
}

```

Figure 330. RmvCon2.java

1. Create the connection specification (Figure 331).

The servlet prepares a specification object identifying information necessary for connecting to the underlying data server. Some of the information is unique to the specific data server, while some is general, applying to any underlying data server. The servlet either prepares the specification only once and uses it for all user requests, or it prepares a new specification for each user request. If a new specification is prepared for each user request, it must be done before step 3 on page 325, which uses the specification. It is usually done in the init method.


```

public void init(ServletConfig sc){
    try{
        super.init(sc);
    }catch (ServletException e){
        e.printStackTrace();
    }
    // URL is
    url0 = "jdbc:db2:sample";    //local DB

    try{
        // *****
        // * STEP 1 *
        // *****
        // Create JDBC connection specification.
        spec0 = new IBMJdbcConnSpec
(poolName,    // pool name from Webmaster
true,        // waitRetry
jdbcDriver,  // Remaining four
url0,        // parameters are
"db2inst1",  // specific for a
"db2inst1"); // JDBC connection.

```

Figure 331. Create JDBC Connection Specification

It is worth noting that different specifications can be used to get connections with different properties. For example, a data server may allow access to certain critical data only if the connection was created with a specific user ID. Therefore, the specification must properly identify the user ID in order to get the suitable connection to the data server.

2. Get a reference to the connection manager (Figure 332).

The servlet gets a reference to the connection manager to communicate with the connection manager. This needs to be done only once in the lifetime of the servlet.

```

// *****
// * STEP 2 *
// *****
// Get a reference to the connection manager.
connMgr = IBMConnMgrUtil.getIBMConnMgr();
}catch(Exception e){
    System.out.println("set connection spec, get connection manager: " +
e.getMessage());
}

} // init

```

Figure 332. Get a Reference to the Connection Manager RmvCon2.java

3. Get a connection manager connection (Figure 333).

The servlet asks the connection manager for a connection to a specific data server using the connection specification prepared in step 1 on page 324. The

connection object returned is from a connection manager pool and is an instance of a class defined in the connection manager APIs. It isn't an object from a class in the API set of the underlying data server. It calls this first connection a connection manager connection, or a CM connection for short. Usually, your servlet gets a CM connection for every user request.

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException{
    IBMJdbcConn cmConn0 = null;    // for IBMConnMgr

    String tbl = null, col = null;
    String colStr = null;
    Object colVal = null;
    String [] qArgs = new String [6];

    MessageFormat mf = null;
    PrintWriter pw = null;
    try{
        res.setContentType("text/html; charset=SJIS");
        pw = res.getWriter();
        pw.println("<HTML>");
        pw.println("<HEAD><TITLE>RmvCon</TITLE></HEAD>");
        pw.println("<BODY>");
        pw.println("<H1>RmvCon Test</H1>");

        pw.println("getIBMConnection<BR>");
        cmConn0 = (IBMJdbcConn) connMgr.getIBMConnection(spec0, "RmvCon");
```

Figure 333. Get a Connection Manager Connection

4. Use the CM connection to access a pre-established data server connection.

The servlet invokes a method on the CM connection returned in step 3, retrieving an object defined in the API set of the underlying data server. Call this object a data server connection (or a data connection for short) to distinguish it from the CM connection. The data connection, unlike the CM connection, is from the underlying data server API set. The data connection is not created for the servlet. The servlet instead uses the pre-established data connection by virtue of owning a CM connection pool from the pool. The data connection will be used for the actual interactions with the data server, using the methods from the underlying data server API set. For example, in the sample servlet code the underlying data server will be a JDBC database and the data connection object will be from the connection class in the JDBC APIs. The JDBC APIs are found in the java.sql package.

```

pw.println("getJdbcConnection<BR>");

    // *****
    // * STEP 5 *   Run DB query (printRows)
    // *****
    try{
Connection con= cmConn0.getJdbcConnection();

```

Figure 334. *get Connection Object of java.sql Package*

5. Interact with the data server (Figure 335).

The servlet interacts with the data server, retrieving data, updating data, and so on, using methods of the data connection object. These methods will be specific to the underlying data server, because the data connection will actually be from the API set of the underlying data server. The data connection for a different underlying data server will have different methods.

```

Statement stmt = con.createStatement();
pw.print("<B>select * from employee </B><BR>");
ResultSet rs=stmt.executeQuery("select * from employee");

```

Figure 335. *Execute SQL Query and Get ResultSet*

Full documentation for the methods are available with the API documentation for the specific data server product. For example, for a JDBC data connection, you will need to look at the documentation for the java.sql package and at any documentation that comes with the JDBC-enabled relational database you are using. If you use the data connection for more than one data server interaction within the same user request, you may want to verify before the additional interactions that your servlet still owns the associated CM connection. To verify that it still owns a connection, the servlet invokes the `verifyIBMConnection()` method that in turn gets the connection manager to check the verify time stamp of the connection. If the servlet still owns the connection, the last-used time stamp is automatically updated to the current time, as part of invoking the `verifyIBMConnection()` method. The servlet then uses the connection to communicate with the data server, confident that the connection will work. When the servlet finishes with the connection, it releases it back to the connection pool. The connection manager sets the in-use flag to false, and sets the verify and last-used time stamps to the current time.

6. Prepare and send the response.

The servlet prepares and returns the response to the user request. In this step you will probably not be using any connection manager APIs.

```

// print out a header row of column names
ResultSetMetaData rsmd = rs.getMetaData();
int numCols = rsmd.getColumnCount();
for(int i = 1;i <= numCols;i++){
    col = rsmd.getColumnName(i);
    pw.print("<B>" + col + " </B>");
}
pw.println("<BR>");
}
rs.close();
stmt.close();

```

Figure 336. *RmvCon2.java* ()

Add a `removeConnection` statement when the exception is caught (Figure 337):

```

}catch(Exception e ){
pw.println("Exception!<BR>");
e.printStackTrace(pw);
//*****
// remove IBMConnection
//*****
try{
    cmConn0.removeIBMConnection();
    pw.println("removeIBMConnection...<BR>");
}catch(Exception e2 ){
    pw.println("Exception at removeIBMConnection<BR>");
    pw.println(e.toString()+"<BR>");
} //end of catch e2
//*****
    }
    pw.println("</BODY>");
    pw.println("</HTML>");
    pw.flush();
}catch( Throwable e ){
    pw.println(e.toString());
    pw.flush();
}
}

```

Figure 337. *RmvCon2.java*

7. Release the connection (Figure 338).

The servlet returns the CM connection to the connection manager pool, freeing the connection for use by another servlet or by another request to the same servlet.

```

// *****
// * STEP 6 *
// *****
// Release the connection back to the pool.
finally{
    if(cmConn0 != null){
try{
    cmConn0.releaseIBMConnection();
    pw.println("cmConn0 released");
}catch(IBMConnMgrException e){
    System.out.println("release connection: " + e.getMessage());
}
    }
}
} // end of doGet()
} //end of class RmvCon2

```

Figure 338. Release the Connection

Connection Manager APIs

You can find the connection manager APIs at:

[http://\[hostname\]/IBMWebAS/doc/apidocs/Package-com.ibm.servlet.connmgr.html](http://[hostname]/IBMWebAS/doc/apidocs/Package-com.ibm.servlet.connmgr.html).

Chapter 6. Enterprise Access

In a conventional enterprise system, relational databases implement enterprise data, whereas transaction processing such as CICS implements enterprise logic. In many cases, this model brings up many issues such as reusability, application complexity, distributed processing, presentation quality and difficulties in serving an emerging service channel such as the Internet. Current Internet technology has enabled enterprises to serve customers, or even to serve other enterprises via business-to-business transaction directly over the Internet. In any case, enterprise logic and data accesses are very crucial.

The first modification to the conventional model is to change the view of enterprise data and logic into an object model. In this model, the object not only represents data but also logic pertaining to the data and relationships to other objects. Next, to improve architectural performance such as distributed computation and better reusability, we extend the object model into a component model. The Java bean is the component model in the Java language. The Enterprise Java Bean is the refinement of the Java bean model to support critical enterprise tasks.

The ultimate goals in using a component model in the enterprise access architecture are:

- Reusability of enterprise computational elements
- Scalability, managing application complexity
- Maintainability of applications
- Separation of responsibility between presentation and content generation (enterprise logic)
- Object persistence, storing object in a database

The EJB model is designed specifically to address these issues. It is the best model in the Java platform for handling enterprise access. The EJB encapsulates enterprise data and associated logic into one component. Since EJB represents data, it should be persisted (stored) into a database. As EJB may need to store its logic state, the persistency means storing the EJB data and logic state into the database. The EJB model delivers two types of persistence:

1. EJB Container Managed Persistence

It provides a simple persistence solution by letting the container handle the persistence. However, it can't handle complex issues such as object associations, inheritance and complex mappings.

2. EJB Bean Managed Persistence

In this EJB type, the EJB manages its own persistence. Any persistence issues should be handled by the EJB.

The EJB model is still undergoing refinement. Some of the current EJB issues are handling complex persistence and better managing object relationships.

In WebSphere Application Server, there are three enterprise access options:

- Using servlets to access enterprise data/logic directly. It is appropriate only for simple applications, as it is not scalable to a complex model and it has no inherent reusability.
- Using Data Beans to handle simple data operations directly. This technique has no object persistence and is not scalable to the complex model.
- Using Enterprise Java Beans would be the logical answer for an enterprise access issue.

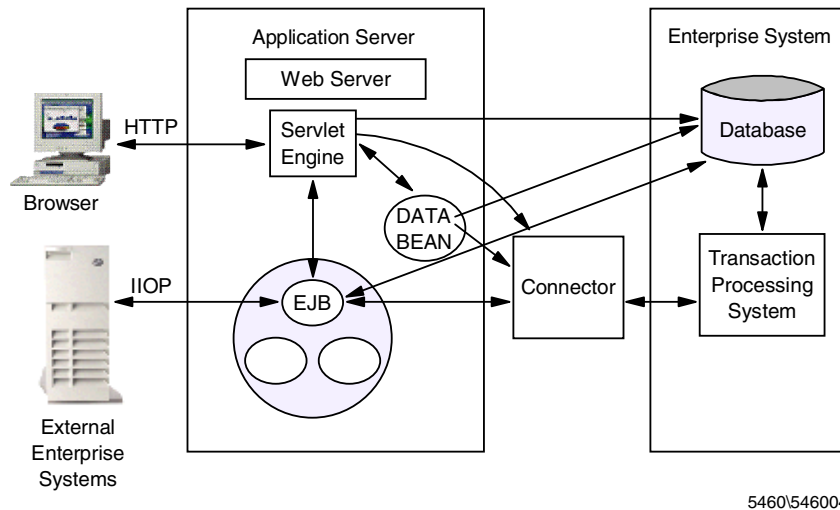


Figure 339. Enterprise Access in the WebSphere Environment

In this chapter, we discuss enterprise access in the WebSphere environment for the following:

- Enterprise data access using a Java component to access an enterprise relational database. We will discuss two underlying Java data access models: the JDBC and the SQLJ.
- Enterprise logic accesses using connectors. This makes use of an existing transaction processing system using the component model. Here, we describe the two best techniques from IBM:
 1. Using MQSeries for exchanging messages on any platform.
 2. Using the CICS connector for accessing CICS servers that serve most enterprise legacy systems.

6.1 JDBC

This section shows how to set up connectivity between WebSphere Application Server and back-end databases.

6.1.1 JDBC Concepts

JDBC is a Java data interface standard that provides access to a wide range of relational databases. To deliver database-independent access in Java language, Java has included JDBC as a standard part of the Java platform. JDBC is a set of interfaces that are implemented by the database vendor, and the vendor implementation of these interfaces is called the JDBC driver for that vendor. The

latest JDBC level is JDBC 2.0, which is available as a separate package from Sun, or as a part of the Java 2 platform. Virtually all database vendors have adapted the JDBC specification into their database products. However, the vendor may not have implemented the latest JDBC version and does not have all driver types as specified in the specification. Contact your database vendor for specific information about its JDBC drivers. To find general information about JDBC products and drivers, go to <http://java.sun.com/products/jdbc>.

JDBC enables Java programs to create sessions to databases, execute SQL statements, and retrieve the results. The JDBC specification delivers a *call-level* SQL interface for Java based on the X/Open SQL call level interface specification. To confine SQL syntax and semantic diversities across database platforms, JDBC also sets minimum SQL conformance to the SQL92 entry level SQL standard. This gives guaranteed wide portability for applications designed to run on many platforms.

The call-level interface limits operations to execute only *raw* SQL statements and to retrieve the results. A Java program can issue an SQL string to be processed by a database server at runtime. This mechanism also means dynamic compilation, privilege and authorization checks. It allows flexibility for the program to construct variable queries that are defined at run time. This model is also known as the dynamic SQL model.

The JDBC specifies four major components: JDBC drivers, connections, statements, and a result set. The database vendors deliver only the driver, which should comply with JDBC specifications. Other components are in the JDBC API package, which is the `java.sql` package. The JDBC API provides interface classes for working with these components:

- The `java.sql.Driver` and `java.sql.DriverManager` for managing JDBC drivers
- The `java.sql.Connection` for using connections
- The `java.sql.Statement`, for constructing and executing SQL statements
- The `java.sql.ResultSet` for processing the results

6.1.1.1 JDBC Driver

JDBC defines standard API calls to a specified JDBC driver. A JDBC driver is a piece of software that performs the actual data interface commands. It is also considered as the lower level JDBC API. This driver has interfaces in the form of database client calls or database network protocol commands that are serviced by a database server.

Depending on the interface type, there are four types of JDBC drivers:

1. Type 1, JDBC-ODBC bridge

The Type 1 driver translates JDBC API calls into ODBC API calls.

2. Type 2, Native-API driver

The Type 2 driver translates JDBC API calls into database native API calls. As the driver uses native APIs, a Type 2 driver is vendor dependent. The driver consists of two parts: a Java language part that performs the translation, and a set of native API libraries.

3. Type 3, Net-Protocol

The Type 3 driver translates JDBC API calls into DBMS-independent network protocol calls. The database server interprets these network protocol calls into specific DBMS operations.

4. Type 4, Native-Protocol

The Type 4 driver translates JDBC API calls into DBMS native network protocol calls. These native protocol calls are then converted by the database server into DBMS operations.

A Java program can create several connections to several different databases using different drivers. To manage driver operations, JDBC provides a driver manager class, the `java.sql.DriverManager`, which is responsible for loading drivers and creating new database connections.

The `DriverManager` registers any JDBC driver that is going to be used. If a Java program issues a JDBC operation on a non-registered driver, JDBC will raise a *No Suitable Driver* exception. There are several ways to register a driver:

- Register the driver explicitly by using:

```
DriverManager.registerDriver(<driver-instance>)
```

where <driver-instance> is an instance of the JDBC driver class.

- Load the driver class by using:

```
Class.forName(<driver-class>)
```

where <driver-class> is the JDBC driver class.

This will load the driver into the Java Virtual Machine. When loaded, each driver must register itself implicitly by using the `DriverManager.registerDriver` method.

For example, to register the DB2 JDBC Type 2 driver which is in the `COM.ibm.db2.jdbc.app` package, you can use either:

```
DriverManager.registerDriver( new COM.ibm.db2.jdbc.app.DB2Driver() );
```

or

```
Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
```

Note: When a driver has a native API part and Java can't find the native API library, JDBC will raise a *No Suitable Driver* condition, even when the `DriverManager` has registered the driver. In WebSphere you should also include the native library directory path into the user `libpath` in the Java Engine setup.

6.1.1.2 JDBC Connection

When a program accesses a database, the program should create a session into the database. This session holds a context for executing SQL statements and returning the results. In the JDBC specification the session is represented by a connection. Therefore, a JDBC connection is a session from a Java program to a database.

A connection is identified by a URL, which has the following format:

```
"jdbc:<subprotocol>:<subname>"
```

where <subprotocol> specifies a particular database connectivity mechanism as supported by a particular driver. The <subname> is a specific parameter of the

subprotocol or driver, and it is dependent on the selected subprotocol. For example, to define a connection to the SAMPLE database:

```
String urlString = String("jdbc:db2:SAMPLE");
```

To open a connection, a Java program should ask the DriverManager to open a connection using a specified protocol and database:

```
Connection aConnection = DriverManager.getConnection(urlString);
```

The DriverManager will ask each driver in its driver list to support the specified protocol. The first driver that can support the protocol is selected and is used to open a connection to the specified database. By specifying different connection URLs, a Java program can create several connections to several databases simultaneously, even with different drivers.

6.1.1.3 JDBC SQL Statement

JDBC allows you to create and execute SQL statements. The SQL statements being used should comply with the SQL92 entry level standard, which is reasonably powerful. The program creates SQL statements and sends them to the database server to be executed in the session context.

The JDBC has three interface classes for SQL statements: the `java.sql.Statement` and its two sub-interfaces: `java.sql.PreparedStatement` and `java.sql.CallableStatement`. Based on these interfaces, there are three alternatives for creating SQL statements:

- Create a statement object by instantiating the `java.sql.Statement` class.

This object initially does not represent any SQL statement. It can construct and execute an SQL statement by calling one of its *execute* methods. For every execute method call, the JDBC driver sends the SQL statement for run-time compilation and execution in the database server. For example:

```
Statement stmt = aConnection.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM EMPLOYEE");
```

- Create a prepared statement object for executing pre-compiled SQL statement by instantiating the `java.sql.PreparedStatement` class.

When created, the object sends the SQL statement for compilation. The compilation result is stored to be used in the subsequent execute methods. This mechanism provides more efficient SQL execution than using a statement object.

A prepared statement allows parameterized SQL statements. The parameter is passed by value (as an *IN* parameter) and identified by ? tokens inside the SQL statement. There are several setter methods, `set<DataType>` (*DataType* is one of the common SQL data types), to set a parameter based on its relative position among other parameters in the SQL statement. The statement is executed when the program calls an execute method on the object. For example:

```
// Select all employees which have salary > 30000.00 and
// bonus < 500.00
PreparedStatement stmt = aConnection.prepareStatement(
"SELECT * FROM EMPLOYEE WHERE SALARY > ? AND BONUS < ?");
stmt.setFloat(1,30000f);
stmt.setFloat(2,500f);
ResultSet rs = stmt.executeQuery();
```

- Create a callable statement object for executing database stored procedures by instantiating the `java.sql.CallableStatement` class.

A callable statement is a type of prepared statement that is used for handling a stored procedure call statement. Its interface is a class that extends the `PreparedStatement` class. When created, the object sends the call statement to the database server for compilation. The compilation result is stored to be used in the subsequent `execute` methods.

A callable statement also allows parameterized stored procedures. In addition to IN parameter passing, the interface also handles OUT parameter passing. The `CallableStatement.registerOutputParameter` method registers the type of parameter in the statement. After execution, use a getter method with the same data type to retrieve the parameter. For example:

```
// Calculate, update and retrieve the new salary
// Call GetNewSalary(name,salary)
// name is IN parameter, salary is OUT parameter
CallableStatement stmt = aConnection.prepareCall("CALL GetNewSalary(?,?)");
stmt.setString(1,"Dilbert");
stmt.registerOutputParameter(2,java.sql.Types.DECIMAL,0);
stmt.executeUpdate();
BigDecimal newSalary = stmt.getBigDecimal(2,0);
```

The statement, prepared statement, and callable statement have several `execute` methods:

- `executeQuery(String aSQLStatement)` for executing a query that returns a single result set.
- `executeUpdate(String aSQLStatement)` for executing a statement that returns no row, such as UPDATE, DELETE and INSERT statements.
- `execute()` or `execute(String aSQLStatement)` for executing a complex or several SQL statements altogether which may return several result sets. Use `getResultSet` and `getMoreResults` methods for obtaining the next result set.

6.1.1.4 JDBC Result Set

The JDBC result set contains a set of rows from the results of executing a query. The `java.sql.ResultSet` interface class implements the result set. A column in a row is specified by a positional index or by a column name. To retrieve a column value, the `ResultSet` class provides several getter methods in the form of `get<DataType>` (where *DataType* is one of common SQL data types). The getter methods accepts column position or column name as its parameter. The `ResultSet.next` method advances at the next row. It returns true if the next row exists; otherwise, it returns false. For example:

```
Statement stmt = aConnection.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM EMPLOYEE");
while (rs.next()) { // Retrieve next record
    String employeeNum = rs.getString(1);
    String firstName = rs.getString("FIRSTNAME");
    float salary = rs.getFloat(3);
    System.out.println("Employee No: " + employeeNum
        + " First name: " + firstName
        + " Salary: " + salary);
}
```

6.1.2 Using JDBC in Java Programs

After discussing JDBC components, we can put these components together in a Java program. There are common steps in implementing JDBC in your Java program:

1. Import JDBC classes. JDBC has become a standard in the Java platform. Its classes are contained in the `java.sql` package, which is in the default JDK/JRE class libraries. To include this package in your Java program, insert the following line at the beginning of your program:

```
import java.sql.*;
```

2. Register the JDBC driver. This needs to be done only once in a program's life time. For example, to register the DB2 Type 2 driver:

```
Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
```

You can put this line inside a static initialization block so that it is executed only once.

3. Create a connection to the database:

```
Connection conn = DriverManager.getConnection("jdbc:db2:SAMPLE");
```

4. Create SQL statements and execute them. Store the result set in `ResultSet`.

5. When processing finishes, do not wait for the Java garbage collection. Close the connection immediately to release valuable database resources:

```
conn.close();
```

6.1.2.1 Transaction Support

All new JDBC connections are in auto-commit mode, which means that each statement is executed as a separate transaction. To create a transaction that consists of several statements, you must disable the auto-commit mode by using the `Connection.setAutoCommit(false)` method. Then the connection will set an implicit transaction up to the point where `Connection.Commit` or `Connection.Rollback` is issued. The `Commit` or `Rollback` also set a new implicit transaction.

```
conn = DriverManager.getConnection("jdbc:db2:SAMPLE");
// Disable the Auto-Commit mode
conn.setAutoCommit(false);
// A new transaction begins ...
...
... Some SQL statements are executed here ....
...
// The end of the first transaction
Connection.commit();
// A new transaction begins
...
...
// The end of the second transaction
Connection.rollback();
```

6.1.2.2 Example: JDBCAccess Utility Class

The following Java class illustrates the concepts we have discussed so far. The `JDBCAccess` class is a utility class for creating a general SQL query. It has three methods for basic JDBC operations: `open`, `query` and `close`. The class registers a JDBC driver in its static initialization block. Any Java program can supply an arbitrary `SELECT` statement into a `query` method.

The query catches an SQL exception if the query condition is not a valid SELECT statement, or if the user does not have necessary privileges. In fact, this shows the disadvantage of using a dynamic SQL statement model. The application can't guarantee that the SQL statement will run.

By default, the class static initializer reads file properties/connection.properties for the driver name. The file must contain a line that specifies the driver name, for example:

```
...
jdbc.driver = COM.ibm.db2.jdbc.app.DB2Driver
...
```

This file can also contain JDBC and SQLJ parameters. In the WebSphere Application Server environment, you should put this file as:

```
<ASROOT>\properties\connection.properties.
```

The class provides three forms of open methods:

1. open(), which does the actual connection creation.
2. open(String url, String user, String password) sets connection parameters and calls open().
3. open(String propertyFileName). To enable easy configuration, this class can read connection parameters from a property file. The file should contain property entries for a URL connection, database user ID and password. For example, a property file might look like the following:

```
#
# Sample property file for JDBC ...
#

jdbc.url = jdbc:db2:SAMPLE
jdbc.user = tsup1
jdbc.password = SWA109R
```

The following lines of codes illustrate the usage of this class:

```
...
JDBCAccess jdbc = new JDBCAccess();

jdbc.open("jdbc:db2:SAMPLE", "db2inst1", "SWA109R");

String qSql = "SELECT firstnme, salary FROM employee";
ResultSet rs = jdbc.query(qSql);

while(rs.next()) {
    System.out.println("First Name: " + rs.getString(1)
        + " Salary: " + rs.getFloat(2));
}
...
```

Later in this chapter, we use this class again for generic JDBC query for DB2 and Oracle platforms.

```

package com.ibm.redbook.sg245460;

import java.sql.*;
import java.util.*;
import java.io.*;

public class JDBCAccess
{
    private Connection conn=null;
    private Statement stmt = null;
    private ResultSet rset = null;
    //
    // Default driver and connection parameters
    //
    private static String driverName = "COM.ibm.db2.jdbc.app.DB2Driver";
    private String urlString = "jdbc:db2:SAMPLE";
    private String dbUser = "tsup1";
    private String dbPassword = "SWA109R";

    static {
        try {
            Properties p = new Properties();

            p.load(new FileInputStream("properties/connection.properties"));
            driverName = p.getProperty("jdbc.driver");
            Class.forName(driverName);

        } catch(IOException e) {
            e.printStackTrace();
        } catch(ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    public Connection open() throws SQLException {
        conn = DriverManager.getConnection (urlString,dbUser,dbPassword);
        return conn;
    }

    public Connection open(String url, String user, String password) throws SQLException {
        urlString = url;
        dbUser = user;
        dbPassword = password;
        return open();
    }

    public Connection open(String propertyFileName) throws SQLException {
        try {
            Properties p = new Properties();
            p.load(new FileInputStream(propertyFileName));

            urlString = p.getProperty("jdbc.url");
            dbUser = p.getProperty("jdbc.user");
            dbPassword = p.getProperty("jdbc.password");

        } catch(IOException e) {
            e.getMessage();
        }
        return open();
    }
}

```

Figure 340. JDBC Example: JDBCAccess Utility Class (1/2)

```

public ResultSet query(String qSql) {
    rset = null;
    try {
        stmt = conn.createStatement ();
        rset = stmt.executeQuery (qSql);
    } catch(SQLException e) {
        e.printStackTrace();
    }
    return rset;
}

public void close() {
    try {
        rset.close();
        stmt.close();
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}

```

Figure 341. JDBC Example: JDBCAccess Utility Class (2/2)

6.1.3 SQLJ

SQLJ is the embedded SQL in Java language. SQLJ statements are similar to the embedded SQL statements in ordinary programming languages, which are precompiled and bound into the database. SQLJ statements are static and based on the ANSI standard SQLJ language syntax. The SQLJ static SQL model is designed to complement the JDBC dynamic SQL model. In a static SQL environment, a program that contains SQL statements can't change SQL statements during run time. The program undergoes pre-compilation, which checks SQL syntax, verifies SQL consistency against database schema, checks data transformation between Java data types and SQL data types. For more information about SQLJ, visit its Web site at <http://www.sqlj.org>.

SQLJ has two components: the translator and the run-time components. Both are written in pure Java. The SQLJ translator performs precompilation and translates embedded SQL statements into SQLJ run-time calls. The SQLJ specification allows the run-time component to perform SQL operations directly into the database or to issue JDBC calls to the database via the JDBC layer.

Developers create embedded SQL programs by creating Java programs with the .sqlj extension. SQLJ programs embed SQL statements among Java statements. A #sql token in the beginning of a statement identifies an SQLJ statement. The SQLJ translator precompiles .sqlj files into .java files and then automatically calls the Java compiler to produce the .class files. When the program runs, it automatically invokes SQLJ run-time components that call JDBC drivers or call native database operations.

6.1.3.1 SQLJ and JDBC Comparison

SQLJ contains static SQL statements; JDBC supports dynamic SQL statements. In a dynamic SQL model, an application may change any SQL statement during run time, hence it is more flexible. Nevertheless, dynamic SQL statements require run-time compilation, which may catch some run time errors. SQLJ pre-compiles SQL statements before the program executes. It enforces strong type checking between Java data types and SQL data types. The pre-compiled static

statements will ensure all SQL statements are verified against database and will run. Thus, it will reduce application maintenance costs.

SQLJ syntax is more concise than JDBC. SQLJ can embed Java variables into SQL statements. JDBC requires separate statements to bind variables using their position or column name. Developers will find it more convenient to use SQLJ statements. Therefore, it may reduce the development costs.

SQLJ binds static statements into a database. The bind process provides privilege and authorization checking. SQLJ separates the package owner from the package runner, providing better data manageability. JDBC programs can't check user privileges until run time.

Most applications require more static statements than dynamic statements. Yet, to exploit JDBC dynamic SQL features, you can combine SQLJ and JDBC together inside one program. Since SQLJ includes JDBC support, the program can create a JDBC connection and use the connection for JDBC dynamic SQL statements, and use the same connection to set a connection context for SQLJ static SQL statements.

6.1.3.2 SQLJ Statement Construct

SQLJ programs embed SQL statements among Java statements. SQLJ statements are not Java statements. To identify SQL statements inside a program, SQLJ statements always begin with #sql. In the pre-compilation, the SQLJ translator will convert each statement that starts with #sql into corresponding Java statements.

The syntax of SQLJ statement is:

```
#sql <sqlj-clauses>;
```

The semicolon (;) ends the SQLJ statement.

There are two SQLJ statement main constructs: SQLJ declarations and executable statements.

SQLJ Declaration

SQLJ implementations use several Java classes. In developing SQLJ programs, you do not write these classes using Java statements directly. Instead SQLJ allows you to write a concise description of the classes using an SQLJ declaration clause. Since a declaration represents a class, the rules for placing a declaration is the same as those of Java classes.

Now there are only two types of SQLJ classes: the connection context classes and the iterator classes. The syntax of a connection context declaration is:

```
#sql <modifier> context <context_class_name>  
    implements <interface_class>  
    with (var1=val1,..);
```

where <modifier> can be any Java class modifier, such as abstract, final, and public. The `implements` clause is optional and is used to derive some interfaces from interface classes. The `with` clause is optional and is used to initialize variables with constants.

The syntax of iterator declaration is:

```
#sql <modifier> iterator <iterator_class_name>
    implements <interface_class>
    with (var1=val1,..)
    (<type_declaration>);
```

where <type_declaration> contains class parameter type declaration.

SQLJ Executable Statement

An SQLJ executable statement represents a block of Java statements. The embedded SQL statement will be translated into a block of corresponding Java statements that perform the actual database calls. In the executable statement, embedded SQL is enclosed with curly brackets. Depending on the SQL statement, an executable statement may or may not return rows of records. An assignment clause has a result expression for returning the result set. The syntax of the executable statements is either in the form of a statement clause:

```
#sql [<context>] { <embedded-sql-statements> };
```

which returns no result, or in the form of an assignment clause:

```
#sql [<context>] <result> = { <embedded-sql-statements> };
```

which has the <result> variable to hold the resulting rows or operation counts. The <context> is optional and specifies either a connection context or an execution context.

Java Host Variable and Host Expression

SQLJ allows Java variables to be included inside SQLJ statements. Such variables are known as host variables. You can also create Java expressions using Host Variables inside SQLJ statements, to form Java host expressions.

A host variable or a host expression begins with a colon (:). The syntax for a Java host expression is:

```
:<mode><a-java-variable>
```

or

```
:<mode>(<a-java-expression>)
```

where <mode> is optional and can be IN, OUT or INOUT (default).

SQLJ Context

An SQLJ context holds the state information for processing and retrieving the result of SQL statements. There are two types of SQLJ contexts: the connection context and the execution context.

1. A *connection context* represents a session into a database. One program may access several databases by using multiple connection contexts. You can use a connection context to perform SQL operations on the corresponding database session. To create a connection context, you should create a new connection context class by using the SQLJ connection context declaration. Then, you create a new instance of the created class.
2. An *execution context* represents an instance of the SQLJ run-time component. By specifying execution contexts, you can use multiple threads, with each execution context using a thread; or you can use different SQL execution flows with separate status information for each execution context. Each connection context has its own default. Therefore, you do not need to specify the

execution context for newly created connections. To create a new execution context, you create a new instance of the `sqlj.runtime.ExecutionContext` class. Then, you can combine this execution context with any connection context in the `<context>` clause of SQLJ executable statements.

Result Set Iterator

The Result Set Iterator is the *cursor* equivalent in SQLJ. To retrieve resulting rows from a query, use a result set iterator object. To create an iterator, create the iterator class using the iterator declaration statement, then declare an object of the class. Use this object to hold the result of the SQLJ assignment clause. This operation, in effect, populates the iterator with the resulting rows from the query. You can then retrieve column values from the iterator object.

To access a column in a row, SQLJ allows an iterator to specify columns by data type or by the name and data type. Based on this criteria, SQLJ differentiates two types of iterator:

1. *Positioned Iterator*. This iterator specifies a column by its relative position in the query and its data type. To retrieve a row, use the `FETCH` executable statement with the iterator as the cursor. The `INTO` clause in the `FETCH` statement retrieves column values into Java host variables. The column position in the `INTO` is the same as what is defined in the iterator declaration. For example:

```
#sql public iterator PosIter (short, String, float);
...
class Staff{
...
public void getStaffInfo() {
...
short id;
String name = String("");
float salary;
PosIter staffCursor = null;
...
#sql staffCursor = { SELECT ID, NAME, SALARY FROM STAFF };
...
while (true) {
#sql FETCH :staffCursor INTO :id, :name, :salary;
if( staffCursor.endFetch() ) break;
System.out.println("Id:" + id
+ " name:" + name
+ " salary:" + salary);
}
...
staffCursor.close();
}
}
```

2. *Named Iterator*. This iterator specifies a column by its name and its data type. To retrieve the next row, use the `next()` method of iterator class. To access a column in a row, use the corresponding accessor method. SQLJ translator automatically adds an accessor method for each column into the iterator class. The accessor method has the same name as the column name and returns the same data type as the column data type. For example:

```
#sql public iterator NamedIter (short id, String name, float salary);
...
```

```

class Staff{
...
public void getStaffInfo() {
...
NamedIter staffCursor = null;
...
#sql staffCursor = { SELECT id, name, salary FROM STAFF };
...
while (staffCursor.next()) {
System.out.println("Id:" + staffCursor.id()
+ " name:" + staffCursor.name()
+ " salary:" + staffCursor.salary());
}
...
staffCursor.close();
}
}

```

6.1.3.3 Compiling SQLJ Programs

SQLJ defines a static SQL model, which precompiles embedded SQL statements inside Java programs. The SQLJ translators perform the precompilation. It translates embedded SQL statements in an .sqlj file into the corresponding Java statements in a .java file. The translator is database vendor dependent. Typically the translator can be called from the system's command line by using the `sqlj` command:

```
sqlj <filename>.sqlj
```

This will generate the corresponding java file that has the same file name but with .java extension: filename.java. Then, this Java file can be compiled using Java's javac compiler to produce the class file.

6.1.3.4 SQLJ Profiles and Customization

The SQLJ translator also generates SQLJ profiles for each connection context class. The profile contains information about embedded statement operations. It is useful for separating vendor-specific operations from generic operations. The *customization* process uses the SQLJ profile for creating a class to support vendor-specific operations.

You can perform another customization for the same application in another database platform by using the same SQLJ profile. The customization never removes SQLJ profile entries; it adds or modifies only entries for the corresponding platform. Using this mechanism SQLJ programs retain *binary compatibility* across the database vendor even when you use vendor-specific operations.

The database vendor may include customization process as an option in the `sqlj` program. Alternatively, the vendor may provide a separate Java program by extending the `sqlj.runtime.profile.util.DataCustomizer` class.

The translator writes the profiles into serialized objects, which can be put in the same application package. The name of profile file is in the following format:

```
<filename>_SJProfile<n>.ser
```

where n is 0,1,2, and so on, depending on how many connection contexts are in the program. For example, if the file name is CustomerBean and the bean has three connection contexts, the profile file name is CustomerBean_SJProfile0.ser, CustomerBean_SJProfile1.ser and CustomerBean_SJProfile2.ser.

Some of the browsers (such as Netscape 4.x) don't support serialized objects. To enable such browsers, the customization process can generate .class file from the .ser file. The class file name usually follows the serialized object file name, replacing the .ser file extension with .class.

6.1.3.5 SQLJ JDBC Interoperability

In some cases, a Java program may require both static and dynamic SQL statements. The program should contain both SQLJ and JDBC statements. To do this, SQLJ provides some JDBC support that performs conversion between a connection context and a connection, and between an iterator and a result set. Using this mechanism, the JDBC connection and the corresponding SQLJ connection context will refer to the same database session.

- To convert an SQLJ connection context into a JDBC connection, use the getConnection method of the connection context class:

```
Connection jdbcConn = sqljContext.getConnection();
```

- To convert a JDBC connection into an SQLJ connection context, use a connection context constructor that takes a JDBC connection as its input parameter and returns the corresponding SQLJ connection context:

```
#sql context SQLJContext;  
SQLJContext sqljContext = new SQLJContext(jdbcConn);
```

- To convert an SQLJ iterator into a JDBC result set, use the getResultSet method of iterator class to return a JDBC result set:

```
SQLJIterator sqljIterator;  
ResultSet jdbcResultSet = sqljIterator.getResultSet();
```

- To convert a JDBC result set into an SQLJ iterator, use a CAST executable statement to populate a named or positional iterator object by result set data:

```
ResultSet jdbcResultSet;  
...  
jdbcResultSet = jdbcStatement.executeQuery("SELECT * FROM EMPLOYEE);  
...  
SQLJIterator sqljIterator;  
#sql sqljIterator = { CAST :jdbcResultSet };  
...
```

6.1.4 Using SQLJ in Java Programs

The SQLJ class libraries are available in an sqlj.runtime package from your database vendor. Some SQLJ implementations require JDBC support as well. Therefore, the program needs to import the JDBC support package.

6.1.4.1 Application Flow

In general, the format of SQLJ programs require the following:

- Import SQLJ run-time classes and JDBC support classes that are required by SQLJ. Put the following lines in the beginning of your program:

```
import sqlj.runtime.*; // SQLJ support package  
import java.sql.*; // For JDBC support
```

- Create a connection to the database. If your SQLJ implementation runs on top of a JDBC driver, you should create a JDBC connection first. This connection will be used for creating the SQLJ connection context. For example, suppose you use the DB2 JDBC Type 2 driver to create a session to the SAMPLE database:

```
...
Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
...
Connection conn = DriverManager.getConnection("jdbc:db2:SAMPLE");
conn.setAutoCommit(false); // disable JDBC Auto-Commit mode
```

- Create an SQLJ connection context class by adding a connection declaration clause:

```
#sql context ConnContext
```

- Create a connection context object by using an existing JDBC connection:

```
ConnContext ctx = new ConnContext(conn);
```

Or, alternatively, if you do not create a JDBC connection, you can specify the connection URL directly:

```
ConnContext ctx = new ConnContext("jdbc:db2:SAMPLE",false);
```

- Create iterator class to hold the result set:

```
#sql iterator PosIter (String, String, float);
```

- Create executable statements. Populate iterator objects with resulting query results:

```
...
PosIter staffCursor;
...
#sql [ctx] staffCursor = { SELECT empno, firstnme, salary FROM EMPLOYEE
};
...
while (true) {
    #sql FETCH :staffCursor INTO :empno, :firstName, :salary;
    if( staffCursor.endFetch() ) break;
    System.out.println("Employee Number:" + empno
        + " First Name:" + firstName
        + " Salary:" + salary);
}
...

```

- Retrieve column values using iterator accessor methods or FETCH statements:

```
...
while (true) {
    #sql FETCH :staffCursor INTO :empno, :firstName, :salary;
    if( staffCursor.endFetch() ) break;
    System.out.println("Employee Number:" + empno
        + " First Name:" + firstName
        + " Salary:" + salary);
}
...

```

- After the program finishes processing the query, close the iterator objects to release valuable database resources:

```
ctx.close();
```

6.1.4.2 Example: ReservationSQL

The following Java class illustrates the concepts we have discussed so far. The ReservationSQL is a utility class for database operations on a reservation table. It encapsulates database operations on the tsup_reservation table.

By default, the class static initializer reads file properties/connection.properties for the driver name. The file must contain a line that specifies the driver name. For example:

```
...
sqlj.driver = COM.ibm.db2.jdbc.app.DB2Driver
...
```

This connection.properties file can also contain JDBC and SQLJ parameters. In the WebSphere Application Server environment, you should put this file as:

```
<ASROOT>/properties/connection.properties.
```

The class provide three forms of open methods:

- open(), which does the actual connection creation.
- open(String url, String user, String password) sets connection parameters and calls open().
- open(String propertyFileName). To enable easy configuration, this class can read connection parameters from a property file. The file should contain property entries for a URL connection, database user ID and password. For example, a property file might look like the following:

```
#
# Sample property file for SQLJC ..
#

sqlj.url = jdbc:db2:@SAMPLE1
sqlj.user = db2inst1
sqlj.password = SWA109R
```

The following lines of codes illustrate the usage of this class:

```
...
ReservationSQL reservation = new ReservationSQL();

reservation.open("properties/connection.properties");

ResultSet rs = reservation.queryAll();

while(rs.next()) {
    System.out.println("Agent Id: " + rs.getString(1)
        + " Ref Id: " + rs.getInt(2)
        + " Package Name: " + rs.getString(3));
}
...
```

Later in this chapter, we use this class in Oracle platforms.

```

package com.ibm.redbook.sg245460;

import java.sql.*;
import oracle.sqlj.runtime.Oracle;
import java.io.*;
import java.util.*;

#sql context ConnContext;
#sql iterator ReservIter(String, int, String);

public class ReservationSQL {

    private static String driverName = "COM.ibm.db2.jdbc.app.DB2Driver";
    private String urlString = "jdbc:db2:SAMPLE";
    private String dbUser = "tsup1";
    private String dbPassword = "SWA109R";

    private Connection jdbcConn;
    private ConnContext sqljConn;
    private ResultSet rs;

    static {
        try {
            Properties p = new Properties();
            p.load(new FileInputStream("properties/connection.properties"));
            driverName = p.getProperty("sqlj.driver");
            Class.forName(driverName);
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    public int open() {
        int rc=0;
        try {
            jdbcConn = DriverManager.getConnection(urlString,dbUser,dbPassword);
            jdbcConn.setAutoCommit(false);
            sqljConn = new ConnContext(jdbcConn);
        } catch (SQLException e) {
            e.printStackTrace();
            rc = e.getErrorCode();
        }
        return -rc;
    }

    public int open(String propertyFileName) {
        try {
            Properties p = new Properties();
            p.load(new FileInputStream(propertyFileName));
            urlString = p.getProperty("sqlj.url");
            dbUser = p.getProperty("sqlj.user");
            dbPassword = p.getProperty("sqlj.password");
        } catch (IOException e) {
            e.printStackTrace();
        }
        return open();
    }

    public int open(String url, String user, String password) {
        urlString = url;
        dbUser = user;
        dbPassword = password;
        return open();
    }
}

```

Figure 342. SQLJ Example: ReservationSQL.sqlj (1/2)


```

public int create(String agentId, int refId, String packageId)
{
    int rc = 0;

    try {
        #sql [sqljConn] { INSERT INTO tsup_reservation
            VALUES (:agentId,:refId,:packageId)};
        #sql [sqljConn] { COMMIT };
    } catch (SQLException e) {
        e.printStackTrace();
        rc = e.getErrorCode();
    }
    return -rc;
}

public ResultSet queryAll() {
    rs = null;
    try {
        ReservIter reservCursor;
        #sql [sqljConn] reservCursor = { SELECT agent_id, ref_id, package_id
            FROM tsup_reservation };

        rs = reservCursor.getResultSet();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return rs;
}

public int delete(String agentId, int refId) {
    int rc=0;
    try {
        #sql [sqljConn] { DELETE FROM tsup_reservation
            WHERE agent_id = :agentId
            AND ref_id = :refId };

        #sql [sqljConn] { COMMIT };
    } catch (SQLException e) {
        e.printStackTrace();
        rc = -e.getErrorCode();
    }
    return rc;
}

public void close() {
    try {
        sqljConn.close();
        jdbcConn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}

```

Figure 343. SQLJ Example: ReservationSQL.sqlj (2/2)

6.2 Using DB2 UDB for WebSphere Applications

WebSphere applications use Java language as the underlying language. To perform database access, the applications require a Java interface to the database. DB2 provides Java support for accessing the database using Java language. DB2 has delivered JDBC drivers and SQLJ support in its product for OS/2, Windows NT, AIX, Solaris, HP/UX, SCO-UNIX, Linux, OS/400 and OS/390 platforms. Although our focus is on Windows NT and AIX Java support, the techniques discussed in this section should also apply to other platforms.

A Java program can access DB2 databases in various ways. In any case, the Java program acts as a client to the DB2 Server component. The program uses the DB2 Client component to interface with the DB2 Server.

You can install the DB2 Server in the same machine as the WebSphere Application Server. However, as the server, the DB2 Server normally resides in

the data server. In typical enterprise environments, the Web server (application server) and the data server are located on different machines. Otherwise, the Web server might overload data server performance. In such cases, you will need a DB2 Client component on the Web server machine and a remote connection between the DB2 Server and the DB2 Client component. To set up a remote DB2 connection, follow the steps described in 2.1.7.3, “Configuring Remote Interface” on page 55.

6.2.1 DB2 Java Support

For Windows NT and AIX, DB2 Java support is available in DB2 UDB, CAE and SDK. The minimum requirements for Java support are DB2 UDB Version 5.0 and IBM JDK 1.1.6. To use JDBC and SQLJ, you need to set up your DB2 environment properly. Follow the steps described in 2.1.7.2, “Setting Up DB2 in WebSphere Environment” on page 53.

6.2.1.1 DB2 JDBC Drivers

DB2 JDBC main usages are for executing dynamic SQL statements and for supporting upper data layers such as SQLJ and IBM DataAccess beans.

DB2 provides two JDBC drivers: a native API Type 2 driver and a net protocol Type 3 driver. In DB2 UDB 5.2, DB2 delivers a set of JDBC 1.1 drivers and another set of JDBC 2.0 drivers. The Java class libraries are in db2java.zip. The native API library is in db2java.dll (NT) or libdb2java.so (AIX).

- The native API driver is for server-side applications that run on the application server. The driver consists of Java parts and a C native API interface (DB2 CLI). The driver translates JDBC calls into DB2 CLI calls. It then gives the calls to the DB2 Client component, which passes them to the DB2 Server. After that, the driver returns the result back to the application. The class name for this JDBC driver is COM.ibm.db2.jdbc.app.DB2Driver, which is in db2java.zip.
- The net protocol driver is for creating an application using the Java applet model. The driver is a 100% Java program that is downloaded along with the applet to the client browser. The driver exchanges messages, by using the TCP socket protocol, to a JDBC Applet Server in the application server machine. The JDBC Applet Server translates applet messages into DB2 CLI calls. It sends the calls into the DB2 Client component, which passes them to the DB2 Server. Then, the JDBC Applet Server passes the results back to the applet. The class name for this JDBC driver is COM.ibm.db2.jdbc.net.DB2Driver, which is in db2java.zip.

The JDBC Applet Server is a stand-alone socket server application. To activate the JDBC Applet Server, enter the following command:

```
db2jstrt <port>
```

This will activate the JDBC Applet Server listening on port <port>.

Connection URL Syntax

To open a database connection, the program should supply a connection URL, database owner, logon user ID and password. A DB2 JDBC connection URL has the following formats:

- Using DB2 native API driver:

```
"jdbc:db2:<database_name>"
```

- Using DB2 net protocol driver:

```
"jdbc:db2://<jdbc_applet_server_hostname>:<port>/<database_name>"
```

For example, to open a connection to the SAMPLE database using the DB2 native API driver, the connection URL is:

```
String urlString = "jdbc:db2:SAMPLE";
```

Or using the DB2 net protocol driver with JDBC Applet Server listening on mylocal.host.com port 9090, the connection URL would be:

```
String urlString = "jdbc:db2://mylocal.host.com:9090/SAMPLE";
```

Other information can be supplied as Java properties to the `DriverManager.getConnection` method. For example, to open a connection using the sDB2 native API to the SAMPLE database owned by db2inst1, log on as user "auser", with password "letmein":

```
Properties infoProp = new Properties();
infoProp.set("owner", "db2inst1");
infoProp.set("userid", "auser");
infoProp.set("password", "letmein");
Connection aConnection = DriverManager.getConnection(urlString, infoProp);
```

For accessing a database in a remote data server, DB2 Client maps the given database name `<database_name>` into a remote database, using its remote interface configuration. To do this, you should configure a remote interface as explained in 2.1.7.3, "Configuring Remote Interface" on page 55. Alternatively, you can supply the data server host name and port number directly to the connection URL:

```
"jdbc:db2://data_server_name:data_server_port/<database_name>"
```

For example, to access the SAMPLE database on remote.host.com using port 9090, the connection URL becomes:

```
String urlString = "jdbc:db2://remote.host.com:9090/SAMPLE"
```

Note that the syntax for the native API JDBC driver accessing a remote database is the same as the net protocol JDBC driver syntax.

6.2.1.2 DB2 SQLJ Support

DB2 SQLJ main usage is for static SQL execution and Java stored procedures. You can use SQLJ in a client-side Java applet or a server-side Java application.

DB2 delivers SQLJ support that complies with the ANSI X.3.135 standard. DB2 SQLJ runs on top of DB2 JDBC drivers. It provides SQLJ Java classes, translator, run-time, and customization tools. The DB2 SQLJ Java libraries are in `sqlj.zip` and `runtime.zip`.

The SQLJ translator will precompile SQLJ file into a Java program. To use the translator, make sure that `sqlj.zip` and `runtime.zip` are in your system classpath, then call `sqlj` program from the command line:

```
sqlj <filename>.sqlj
```

This will create a Java program with the same name, <filename>.java and serialized SQLJ profiles <filename>_SJProfilen.ser (n=0,1,2,..., for each connection context) for customization purposes.

DB2 SQLJ provides a batch file, embprep, to customize SQLJ profiles. The batch file calls a Java application, sqlj.runtime.profile.util.DataCustomizer using user-supplied parameters. To use the customizer, call embprep from the command line:

```
embprep <filename> <database_name> <userid> <password> ...
```

This will create a customization class, <filename>_SJProfileKey.class, which contains information about operations performed by embedded statements. Practically, the embprep will bind the program into the database. The SQLJ profiles contains the binding information. The embprep also verifies:

- SQL consistency against database schema
- User privilege for using database resources

Note that the binding process requires DB2 SQLJ programs file name <filename> which should be fewer than eight characters.

To run embprep, you should have privileges to perform all operations in embedded statements. Figure 344 summarizes the steps for compiling SQLJ programs.

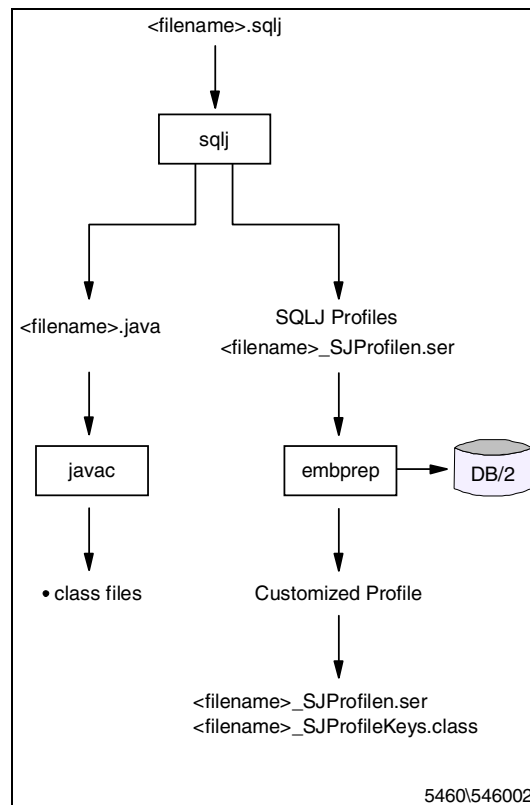


Figure 344. Preparing Programs Written in SQLJ Language

6.2.2 Setting Up DB2 Java Support for the WebSphere Environment

The only requirement for using Oracle Java support in the WebSphere environment is to set up the Java path information in the WebSphere Java Engine setup.

6.2.2.1 Setting Up System CLASSPATH Variable

Setting Java library paths into system environment variables may be useful for developing and debugging purposes. Add the following path into the CLASSPATH variable:

```
<INSTHOME>/sqllib/java/db2java.zip  
<INSTHOME>/sqllib/java/sqlj.zip  
<INSTHOME>/sqllib/java/runtime.zip
```

where <INSTHOME> is your DB2 instance directory.

6.2.2.2 Setting Up WebSphere CLASSPATH and User LibPath

In the WebSphere Administration, you should also include Java paths in the Application Server classpath and the user libpath.

- Log on to WebSphere Administration Tool using the administrator user ID.
- Select **Setup > Java Engine** to bring up the Java Engine setup form.
- On the Path tab, add the following Application Server Classpath entry field:

```
<INSTHOME>/sqllib/java/db2java.zip  
<INSTHOME>/sqllib/java/sqlj.zip  
<INSTHOME>/sqllib/java/runtime.zip
```

- You should also add application SQLJ profiles to the Application Server Classpath entry field, so that they can be found by the SQLJ run-time component.
- For the Windows NT platform, add <INSTHOME>\sqllib\bin in the user libpath entry field. For the AIX platform, add <INSTHOME>/sqllib/lib in the user libpath.

After changing this path information, restart both the Web server and WebSphere. In AIX, you may have to kill the WebSphere processes before restarting the Web server.

6.2.3 DB2 Java Examples

In this section, we give two examples of using DB2 Java support in the WebSphere environment. These examples are deployed as Java Beans.

6.2.3.1 The XtremeTravel Inc.

We will develop our DB2 Java examples based on the XtremeTravel example, which comes with WebSphere Application Server 2.0. We will add the following functions in the original XtremeTravel:

- Customer registration
- Customer logon

6.2.3.2 DB2, JDBC, and SQLJ Example: CustBean.sqlj

The following example is a data bean. It contains both JDBC and SQLJ statements in one program. The SQLJ statements make use of the JDBC connection as SQLJ connection context.

The SQLJ static INSERT statement inserts customer registration data into a customer table. The program uses the JDBCAccess utility class to create a connection and perform a query. A query on the customer table does the logon verification by checking the customer table for a particular user ID and password combination.

The bean uses a property file, which is <ASROOT>properties/connection.properties, to set up the JDBC connection property.

```
# WebSphere Application Server
# IBM Redbook SG245460
#
# Sample property file for JDBC and SQLJ
#
#Wed May 12 17:36:02 EDT 1999

jdbc.driver = COM.ibm.db2.jdbc.app.DB2Driver
jdbc.url = jdbc:db2:xtcust
jdbc.user = db2inst1
jdbc.password = SWA109R

...
...
...
```

Figure 345. JDBC Connection Properties in connection.properties File

Figure 346 and Figure 347 show the code listing of CustBean.sqlj:

```

package com.ibm.sg245460.xtreme;

import java.util.*;
import java.io.*;
import java.sql.*;
import sqlj.runtime.*;
import sqlj.runtime.ref.*;
import com.ibm.redbook.sg245460.*;

#sql context ConnContext;

public class CustBean extends java.lang.Object implements java.io.Serializable {

    protected int    custNo = 0;
    protected String title = null;
    protected String firstName = null;
    protected String lastName = null;
    protected String addrLine1 = null;
    protected String addrLine2 = null;
    protected String city = null;
    protected String state = null;
    protected String zip = null;
    protected String country = null;
    protected String userId = null;
    protected String password = null;

    public String getTitle() { return title; }
    public void setTitle(String value) { this.title = value; }
    public String getFirstName() { return firstName; }
    public void setFirstName(String value) { this.firstName = value; }
    public String getLastName() { return lastName; }
    public void setLastName(String value) { this.lastName = value; }
    public String getAddrLine1() { return addrLine1; }
    public void setAddrLine1(String value) { this.addrLine1 = value; }
    public String getAddrLine2() { return addrLine2; }
    public void setAddrLine2(String value) { this.addrLine2 = value; }
    public String getCity() { return city; }
    public void setCity(String value) { this.city = value; }
    public String getState() { return state; }
    public void setState(String value) { this.state = value; }
    public String getZip() { return zip; }
    public void setZip(String value) { this.zip = value; }
    public String getCountry() { return country; }
    public void setCountry(String value) { this.country = value; }

    public void setUserId(String value) { this.userId = value; }
    public void setPassword(String value) { this.password = value; }

    private Connection jdbcConn = null;
    private PreparedStatement stmt = null;
    private ResultSet rs = null;
    private JDBCAccess jdbc = null;
    private ConnContext sqljConn = null;

    public void connectDB() {
        try {
            jdbc = new JDBCAccess();
            jdbcConn = jdbc.open("properties/connection.properties");

            sqljConn = new ConnContext(jdbcConn);

        } catch(SQLException e) {
            e.printStackTrace();
        }
    }
}

```

Figure 346. DB2, JDBC, and SQLJ Example: CustBean.sqlj (1/2)

```

public void registerCust() {
    try {
        #sql [sqljConn] { SELECT MAX(cust_no) INTO :custNo FROM customer };
        custNo++;
        #sql [sqljConn] { INSERT INTO customer (cust_no,title,first_name,last_name,
            addr_line1,addr_line2,city,state,zip,country,
            userid, password)
            VALUES (:custNo, :title, :firstName, :lastName,
                :addrLine1, :addrLine2, :city, :state, :zip, :country,
                :userId, :password) };
        #sql [sqljConn] { COMMIT };
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public int logon() {
    int rc = 0;
    try {
        String qSql = "SELECT cust_no FROM customer ";
        qSql += "WHERE userid = '" + userId + "' AND password = '" + password + "'";
        rs = jdbc.query(qSql);

        if(rs.next()) rc = 0;
        else rc = -1;

    } catch (SQLException e) {
        e.printStackTrace();
    }
    return rc;
}

public void closeDB() {
    jdbc.close();
}

```

Figure 347. DB2 JDBC and SQLJ Example: CustBean.sqlj (2/2)

6.3 Using Oracle for WebSphere Applications

As Java is becoming a universal computing platform, Oracle is integrating Java support in its products. Oracle comes with the Oracle 8i RDBMS server, the JServer and the JDeveloper. Unlike other Oracle database products, the Oracle 8i integrates Java support inside its database server. The JServer is the Oracle Java engine that is fully integrated into Oracle 8i. Oracle also provides JDeveloper for developing Java database applications. These Oracle products allow you to implement EJB, CORBA, and general Web applications. For more information about Oracle Java technology, see <http://www.oracle.com>.

6.3.1 Oracle Java Support

The JServer includes a JVM, JDBC drivers, SQLJ support, an EJB server, JSP and CORBA support. The JVM, which is also known as the Aurora JVM, complies with JDK 1.1.6 specifications. The JVM and other Java support components run within the same address space as the database server. Oracle JDBC Server, SQLJ run time, and EJB Server run on top of the JVM within the database server address space. This mechanism ensures optimum performance for Java database operations.

6.3.1.1 Oracle JDBC Drivers

Oracle provides three JDBC drivers: JDBC OCI driver, JDBC Server driver, and JDBC Thin drivers. In Oracle 8i Version 8.1, all drivers comply with the JDBC 1.2.2 specifications. The JDBC OCI and JDBC Thin drivers are also available in

previous Oracle database server products. The class name for all JDBC driver types is `oracle.jdbc.driver.OracleDriver`. Oracle JDBC support libraries are in `<ORACLE_HOME>/jdbc/lib/classes111.zip`.

The Oracle JDBC OCI driver is a client-side native API driver. It is intended for Java applications or Java programs on an application server. The driver is written in Java and C. It uses C calls to Oracle Call Interface (OCI), which is the native API for accessing the Oracle database.

The Oracle JDBC Server driver is a server-side native API driver. It is intended for Java programs that run inside Oracle JVM. This driver uses the same address space as the database engine.

The Oracle JDBC Thin driver is a native protocol JDBC driver. It enables Java applets to access the Oracle database. It is a 100% Java program. In the Java applet model, the driver is downloaded along with the applet to the client Web browser. When the applet starts, the driver creates a connection to the database server by using the TCP socket protocol. The interaction between the driver and the server uses the Oracle Net8 protocol as the native protocol. Unlike the stateless HTTP protocol, the Thin driver connection is stateful. The Oracle server manages these connections and their states.

Connection URL Syntax

To open a connection to a database, the program should supply a connection URL, logon user ID and a password. For Oracle JDBC OCI and JDBC Server drivers, the URL format is:

```
"jdbc:oracle:oci8:@<database_name>"
```

For the Oracle Thin driver, the URL format is:

```
"jdbc:oracle:thin:@<listener_hostname>:<listener_port>:<database_name>"
```

For example, the URL string for opening a connection to the SAMPLE database, which has the system identifier sam, using JDBC OCI or JDBC Server driver, is the following:

```
String urlString = "jdbc:oracle:oci8:@SAMPLE";
```

Or if you use the JDBC Thin driver with the TCP listener on `mylocal.host.com` port 9090, the URL string is:

```
String urlString = "jdbc:oracle:thin:@mylocal.host.com:9090:SAMPLE";
```

To open a connection, the program should also supply the logon user ID and password. For example, using user ID "scott" with password "tiger":

```
Connection aConnection =  
DriverManager.getConnection(urlString, "scott", "tiger");
```

For accessing a database in a remote data server, you can specify the database location by the Net8 name-value pair:

```
(description=(address=(host=<hostname>) (protocol=tcp) (port=<port>)) (connect_data=(sid=<sid>)))
```

where `<sid>` is database System Identifier (SID).

For example, to connect to the SAMPLE database which has the SID "sam" in a remote data server, remote.host.com, that listens on port 9090, use the following:

```
String urlString = "jdbc:oracle:oci8:@  
(description=(address=(host=remote.host.com) (protocol=tcp) (port=9090))  
(connect_data=(sid=sam))) "
```

6.3.1.2 Oracle SQLJ Support

Oracle also delivers ANSI X3.135 SQLJ supports. Oracle SQLJ runs on top of any Oracle JDBC driver type. The Oracle SQLJ is the Java equivalent of the Oracle Pro*C embedded SQL environment. It enables developers to create precompiled static SQL statements and Java stored procedures.

Oracle SQLJ support provides SQLJ Java classes, translator, run time, and customization tools. To precompile SQLJ programs using the command line, you should include the Java library path into session CLASSPATH (see 6.3.2.1, "Setting Up System CLASSPATH Variable" on page 359).

To precompile the SQLJ translator:

```
sqlj <filename.sqlj>
```

This will generate a Java source program, filename.java, and SQLJ profiles for each connection context in the program. It will also automatically call Java compiler to create Java class files.

6.3.1.3 Oracle Java Stored Procedures

An Oracle Java stored procedure is the PL/SQL equivalent in the Java language. The Java stored procedures are Java programs that are stored in the database, similar to PL/SQL stored procedures. A Java stored procedure uses JDBC or SQLJ to access the database. It has a call specification that is also kept in the database. A PL/SQL program can call a Java procedure by using its call specification. The called Java procedure will be loaded and executed in the Oracle JVM. On the other hand, a Java stored procedure can call a PL/SQL procedure by using a JDBC callable statement. Similarly, other Java programs can also call a Java stored procedure by using a JDBC callable statement.

6.3.1.4 Oracle EJB Server

Oracle has its own EJB Server, which complies with EJB 1.0 specification. Now, Oracle 8i EJB Server does not support entity beans, which is not mandatory in the EJB 1.0 specification. It supports only session beans. A session bean is inherently non-persistent and does not represent business data. Therefore, if persistence is required, a session bean should store its state in the database or access persistent data directly in the database by using JDBC, SQLJ or Java stored procedures.

6.3.1.5 Oracle CORBA Support

As a good alternative, Jserver allows you to use CORBA applications to access the Oracle database. Jserver provides Java classes to build Java CORBA objects and runs them in the database server. Since CORBA has its own set of services, Jserver provides interfaces to bridge Java services or database services into CORBA services, such as:

- Interface from Java Transaction Service (JTS) to CORBA Transaction Service

- JNDI interface into CORBA CosNaming services for storing and retrieving objects to/from Oracle databases

6.3.2 Setting Up Oracle Java Support for the WebSphere Environment

The only requirement for using Oracle Java support in the WebSphere environment is to set up Java path information in the WebSphere Java Engine setup.

6.3.2.1 Setting Up System CLASSPATH Variable

Setting Java library paths into system environment variables may be useful for developing and debugging purposes. Add the following path into the CLASSPATH variable:

```
<ORACLE_HOME>/jdbc/lib/classes111.zip
<ORACLE_HOME>/sqlj/lib/translator.zip
<ORACLE_HOME>/sqlj/lib/runtime.zip
```

where <ORACLE_HOME> is your Oracle instance directory.

6.3.2.2 Setting Up WebSphere CLASSPATH and User LibPath

In the WebSphere Administration, you should also include Java paths into the Application Server classpath and the user libpath.

- Log on to WebSphere Administration Tool using the administrator user ID.
- Select **Setup > Java Engine** to bring up the Java Engine setup form.
- On the Path tab, add the following Application Server Classpath entry field:

```
<ORACLE_HOME>/jdbc/lib/classes111.zip
<ORACLE_HOME>/sqlj/lib/translator.zip
<ORACLE_HOME>/sqlj/lib/runtime.zip
```

- You should also add application SQLJ profiles to the Application Server Classpath entry field, so that they can be found by the SQLJ run-time component.
- Add <ORACLE_HOME>/bin in the user libpath entry field.

After changing this path information, restart both the Web server and WebSphere. In AIX, you may have to kill the WebSphere processes before restarting the Web server.

6.3.3 Oracle Java Examples

In this section, we describe two examples of using Oracle Java support in the WebSphere environment. These examples, one for JDBC and one for SQLJ, are deployed as EJBs.

6.3.3.1 The TravelSupplier Inc.

For our example, assume that there is a fictitious travel company, The TravelSupplier Inc., which supplies travel packages for other travel agencies or for online travel sites, such as our XtremeTravel. The TravelSupplier Inc. has a WebSphere Application server to provide travel data and reservation services for external entities by using EJB. The company uses Oracle 8i Version 8.1.5 as its enterprise database.

This company application on WebSphere delivers two services:

1. Retrieves a list of available travel packages. As the query will be dynamic by nature, we use the JDBC technique to access the database.
2. Performs customer reservation. The reservation request comes from other agencies such as XtremeTravel. In this case, a servlet in XtremeTravel calls the reservation EJB in The TravelSupplier. As the transactions are the same for all travel agencies, we use SQLJ to insert a reservation record into the database.

As a matter of fact, the company has decided to concentrate on its core business, as a wholesale supplier creating attractive travel packages. It does not have and does not want to provide any Web site for individual customers to access. Instead, it makes use of other online travel sites such as XtremeTravel to market its travel packages.

6.3.3.2 Oracle JDBC Example: TravelPackageEJB

The TravelPackageEJB retrieves travel package records from the `tsup_package` table in the `SAMPLE1` database. To deploy this EJB, follow the instructions in 4.2.3, “Deploying an EJB” on page 192. The EJB can be called by an applet, a servlet or other Java beans using the EJB calling mechanism (see 4.3, “Coding WebSphere EJB Clients” on page 207).

Figure 349 shows the bean implementation for TravelPackageEJB. In this EJB, we use the JDBCAccess class as described in 6.1.2.2, “Example: JDBCAccess Utility Class” on page 337. The class will require a property file `<ASROOT>properties/connection.properties` with the following contents:

```
# WebSphere Application Server
# IBM Redbook SG245460
#
# Sample property file for JDBC and SQLJ
#
#Wed May 12 17:36:02 EDT 1999

jdbc.driver = oracle.jdbc.driver.OracleDriver
jdbc.url = jdbc:oracle:oci8:@SAMPLE1
jdbc.user = tsup1
jdbc.password = SWA109R

...
... SQLJ parameters removed for clarity ...
...
```

Figure 348. JDBC Connection Properties in `connection.properties` File

```

package com.ibm.redbook.sg245460.tsup;

import java.rmi.RemoteException;
import java.security.Identity;
import java.util.Properties;
import javax.ejb.*;
import java.lang.*;
import java.sql.*;
import java.util.*;
import com.ibm.redbook.sg245460.*;

public class TravelPackageEJBBean implements javax.ejb.SessionBean {

    private javax.ejb.SessionContext mySessionCtx = null;

    public Vector getPackage() {

        Vector allPackage = null;
        TravelPackage pkg = null;

        ResultSet rs=null;

        try {
            allPackage = new Vector();

            JDBCAccess jdbc = new JDBCAccess();

            jdbc.open("properties/connection.properties");

            rs = jdbc.query("SELECT package_id, package_name, destination, duration, price, start_date FROM tsup_package");

            while(rs.next()) {
                pkg = new TravelPackage();
                pkg.setPackageId(rs.getString(1));
                pkg.setPackageName(rs.getString(2));
                pkg.setDestination(rs.getString(3));
                pkg.setDuration(rs.getShort(4));
                pkg.setPrice(rs.getFloat(5));
                pkg.setStartDate(rs.getDate(6));
                allPackage.addElement(pkg);
            };
            jdbc.close();

        } catch (SQLException e) {
            e.printStackTrace();
        }
        return allPackage;
    } // end of getPackage

    public void ejbActivate() throws java.rmi.RemoteException {
    }
    public void ejbCreate() {
    }
    public void ejbPassivate() throws java.rmi.RemoteException {
    }
    public void ejbRemove() throws java.rmi.RemoteException {
    }
    public void setSessionContext(javax.ejb.SessionContext ctx) throws java.rmi.RemoteException {
        mySessionCtx = ctx;
    }

} // end of TravelPackageEJBBean

```

Figure 349. Oracle JDBC: TravelPackageEJBBean Using JDBCAccess Utility Class

6.3.3.3 Oracle SQLJ Example: ReservationEJB

The ReservationEJB inserts a reservation record into the tsup_reservation table in the SAMPLE1 database. To deploy this EJB, follow the instructions in 4.2.3, “Deploying an EJB” on page 192. The EJB can be called by an applet, a servlet or other Java beans using the EJB calling mechanism (see 4.3, “Coding WebSphere EJB Clients” on page 207).

The ReservationEJB uses ReservationSQL.sqlj, as described in 6.1.4.2, “Example: ReservationSQL” on page 347.

To configure SQLJ parameters, we use the same property file as used by TravelPackageEJB. The property file is <ASROOT> properties/connection.properties, with the following contents:

```
# WebSphere Application Server
# IBM Redbook SG245460
#
# Sample property file for JDBC and SQLJ
#
#Wed May 12 17:36:02 EDT 1999

...
... JDBC parameters removed for clarity ...
...
sqlj.driver = oracle.jdbc.driver.OracleDriver
sqlj.url = jdbc:oracle:oci8:@SAMPLE1
sqlj.user = tsup1
sqlj.password = SWA109R
```

Figure 350. SQLJ Connection Properties in connection.properties File

Figure 351 shows the bean implementation for ReservationEJB:

```
package com.ibm.redbook.sg245460.tsup;

import java.rmi.RemoteException;
import java.security.Identity;
import java.util.Properties;
import javax.ejb.*;
import java.lang.*;
import com.ibm.redbook.sg245460.*;

public class ReservationEJBBean implements javax.ejb.SessionBean {

    private javax.ejb.SessionContext mySessionCtx = null;

    public int createReservation(String agentId, int refId, String packageId) {
        int rc=0;
        ReservationSQL r;
        r = new ReservationSQL();
        rc = r.open("properties/connection.properties");
        if( rc < 0 ) return rc;
        rc = r.create(agentId,refId,packageId);
        return rc;
    }

    public void ejbActivate() throws java.rmi.RemoteException {
    }
    public void ejbCreate() {
    }
}
```

Figure 351. Oracle SQLJ Example: ReservationEJBBean

6.4 Using MQSeries for WebSphere Applications

IBM MQSeries is a message-oriented middleware that enables business applications to exchange messages across various platforms. It provides an easy- to-use common API, called Message Queue Interface (MQI), which encapsulates all complexities in handling message integrity and security and managing underlying communication protocol. The MQI ensures rapid application integration across different platforms. MQSeries is available in all commercial

operating systems. To find more information about IBM MQSeries, visit <http://www.software.ibm.com/ts/mqseries>.

The best way to use MQSeries in the WebSphere Application Server environment is by interfacing MQSeries with Java language. With this strategy, a Java program can exchange messages with the back-end system using MQSeries. Indeed, MQSeries has provided Java support in its MQSeries for Java component. This extends the reach of MQSeries into the Internet. Typical usage of this component in the WebSphere environment is in EJBs for encapsulating enterprise transactions delivered via MQSeries.

In this section, we focus on implementing MQSeries applications with MQSeries for Java. We start by providing a brief overview of MQSeries, before using it with MQSeries for Java. For more information about developing application on MQSeries, read *MQSeries Application Programming Guide*, SC33-0807 and *MQSeries using Java*, SC34-5456. Both documents are also available as online documentations from the MQSeries installation.

6.4.1 MQSeries Overview

MQSeries exchanges information in the form of messages. Applications put and retrieve messages onto and from queues. A queue manager exists to manage queue operations. Each queue manager owns a set of local queues. An application sends a message by connecting to a queue manager and writing the message into a remote queue. The queue manager is responsible for transferring the message via channels to a remote queue manager that owns the remote queue. A remote application retrieves the message from the remote queue. In MQSeries, queue managers, queues, and channels are recoverable resources, which are also known as MQSeries objects.

Messages

MQSeries messages consist of application data and message descriptors. The application data format and application level protocol are defined by the communicating parties. The message descriptor contains the message identification, message type, priority, addressing information, and other control information.

Queues

An MQSeries queue stores messages. A queue is owned by a queue manager. It is implemented in main memory, disk or other auxiliary storage. It has the ability to perform recovery in case of storage failure.

Queue Managers

An application that uses MQSeries facilities should connect to a queue manager, which is also called the application local queue manager. A queue manager owns a set of local queues. A local queue manager's queue is also an application local queue.

A remote queue manager is another application's queue manager. A remote queue is a remote queue manager's queue. An application can put messages into local queues or remote queues. An application can retrieve messages only from local queues.

A queue manager has three types of local queues:

1. Transmission queues to store messages temporarily until they are transmitted successfully to remote queues
2. A dead letter queue to store messages that can't be delivered
3. Application queues to store received messages.

The queue manager keeps definitions for remote queues. This local definition will represent the remote queue. Any operation placed into this definition will be executed as an operation in the remote queue. Therefore, to write to a remote queue, an application puts the message to the definition of the remote queue. The queue manager will automatically transfer the message to the remote queue.

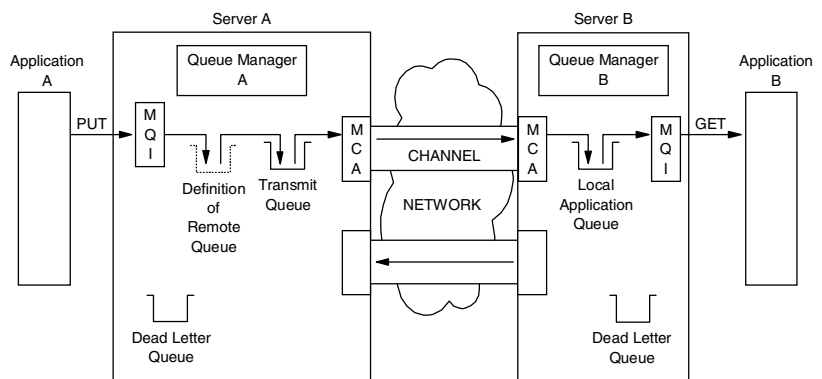
Channel

In MQSeries, a message channel is a communication path between two queue managers. A message channel is a one-way link. Two-way communication needs two message channels. At both ends of a channel, there is a Message Channel Agent (MCA), which is a software component that handles the sending and receiving of messages. Each channel has a sender channel definition and a receiver channel definition. The two channel definitions have the same name.

Addressing Information

An application can exchange messages with another application in the same queue manager. Both applications can access a local queue. In this case, the addressing information is only the local queue name.

If an application sends messages to a remote queue, the application should specify the local definition of the remote queue. MQSeries configuration maps this definition into remote the queue manager, remote queue and transmission queue. Any message put into this definition will be transferred to the remote queue.



5460/546005

Figure 352. MQSeries Concept

Message Queuing Interface (MQI)

MQI is a procedural API for accessing MQSeries facilities. MQI consists of a set of procedures or functions, which are also known as *calls*. Table 32 lists commonly used calls.

Table 32. MQI Calls

Call Verb	Description
MQCONN	Connect an application to a queue manager
MQDISC	Disconnect an application from a queue manager
MQOPEN	Open an object, such as queue
MQCLOSE	Close an object
MQPUT	Put a message into queue
MQGET	Retrieve messages from queue
MQBEGIN	Start a global unit of work
MQCMIT	Commit changes and end a unit of work
MQBACK	Roll back changes and end a unit of work

6.4.2 MQSeries for Java

MQSeries for Java can deliver industrial-strength, message-oriented middleware into an Internet/intranet environment. It consists of a set of Java classes and a native class implementation library.

The minimum software level requirements to use MQSeries are JDK 1.1, MQSeries Server 5.0 and MQSeries Client 2.0. You can obtain MQSeries for Java from the MQSeries Server CD-ROM. You can also obtain MQSeries Client for Java from MQSeries Client CD-ROM. When the installation program asks you to choose MQSeries components, select MQSeries Client for Java (under MQSeries Clients options) and MQSeries Bindings for Java.

In our examples, we use MQSeries Server for Windows NT 5.0, and MQSeries Clients for Windows NT 2.0 and JDK 1.1.6.

6.4.2.1 Setting Up MQSeries for Java

For using MQSeries messaging facilities from the Java environment, you should configure class paths and library paths both in the system environment and in the WebSphere environment. After that, define and set up MQSeries objects that will be used in the applications.

Setting Up System CLASSPATH and Library Path Variables

Placing Java library paths into system environment variables may be useful for developing and debugging purposes. Add the following path into the CLASSPATH variable:

```
<MQROOT>/java/lib
```

Add the following path into the library path variable:

```
<MQROOT>/java/lib
```

where <MQROOT> is the MQSeries root installation directory. For example, if you install MQSeries under d:\mqm then the path will be d:\mqm\java\lib.

Setting Up WebSphere CLASSPATH and User LibPath

In the WebSphere Administration, you should also include this directory in the application server classpath and user libpath.

- Log on to the WebSphere Administration Tool using the administrator user ID.
- Select **Setup > Java Engine** to bring up the Java Engine setup form.
- On the Path tab add <MQROOT>/java/lib to the application server classpath entry field.
- Add <MQROOT>/java/lib to the user libpath entry field.

After changing this path information, restart both the Web server and WebSphere. In AIX, you may have to kill the WebSphere processes before restarting the Web server.

Setting Up Communications

MQSeries provides supports for TCP/IP, LU6.2, NetBIOS and SPX protocols. Before configuring MQSeries, communication links should exist between the server machines using one of the protocols. For TCP/IP, the configuration requires host names and port numbers. Verify that the Domain Name Server works properly and the port numbers are not used by other applications.

Configuring Queue Managers

You can configure MQSeries according to an application’s messaging requirements. MQSeries provides two command sets for performing administration tasks: the Control commands and the MQSC commands. In this section, we use only some basic commands. You can see more commands and their descriptions in the *MQSeries Administration Manual*, which is also available in the MQSeries online documentation.

In our example, we will be connecting an EJB in the application server to a back-end application in MQSeries server-to-server mode. Both the application server machine and the back-end machine have their own MQSeries servers. Table 33 on page 366 and Figure 353 summarize the configuration for both servers:

Table 33. MQSeries Configuration at the Application Server and the Back-End System

Parameter	Application Server	Back-End Application
Hostname(port)	wsnt00(9101)	wtr05100(9102)
Queue Manager	bean.queue.manager	backend.queue.manager
Local Queue	BEAN.QUEUE	BACKEND.QUEUE
Remote Queue Definition	DEF.OF.BACKEND.QUEUE	DEF.OF.BEAN.QUEUE
Transmission Queue	BEAN.TRANSMIT.QUEUE	BACKEND.TRANSMIT.QUEUE
Channel (Sender)	BEAN.CHANNEL	BACKEND.CHANNEL
Channel (Receiver)	BACKEND.CHANNEL	BEAN.CHANNEL

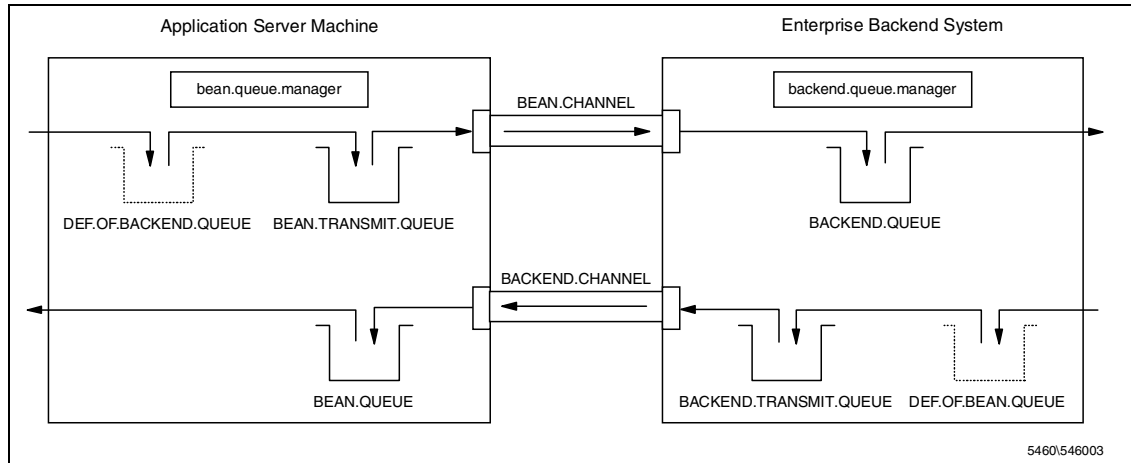


Figure 353. Connection between an Application Server and Back-End System

An MQSeries server contains several queue managers. To use MQSeries messaging, applications must connect to a queue manager. You can create a new queue manager or use the existing one. Before you configure a queue manager, make sure that the queue manager is running.

For example, in the Application Server MQSeries:

- Create the queue manager by using `crtmqm` from the system console:

```
crtmqm bean.queue.manager
```

- Start the queue manager by using `strmqm` from the system console:

```
strmqm bean.queue.manager
```

- Configure the queue manager by using `runmqsc` from the system console:

```
runmqsc bean.queue.manager
```

This command will invoke a console for setting the queue manager parameter. As the `runmqsc` has no prompt, issue the command at the next available line.

- Create the local queue by issuing the following command:

```
define qlocal (BEAN.QUEUE)
```

- Create the local queue for the transmission queue:

```
define qlocal (BEAN.TRANSMIT.QUEUE) usage(xmitq)
```

- Create a local definition of the remote queue. This local definition will represent the remote queue. Any operation of this definition will be executed as an operation in the remote queue. In this case the `BACKEND.QUEUE` is the remote queue at the remote back-end queue manager. The message transfer is done through the local transmission queue:

```
define qremote (DEF.OF.BACKEND.QUEUE) rname (BACKEND.QUEUE) +
rqnname ('backend.queue.manager') xmitq (BEAN.TRANSMIT.QUEUE)
```

- Create a channel for sending messages using TCP/IP protocol to remote machine `wtr05100` on port `9102`:

```
define channel (BEAN.CHANNEL) chltype (SDR) trptype (TCP) +
conname ('wtr05100 (9102)') xmitq (BEAN.TRANSMIT.QUEUE)
```

- Create a channel for receiving messages:

```
define channel(BACKEND.CHANNEL) chltype(RCVR) trptype(TCP)
```

- Exit from runmqsc session:

```
end
```

At the back-end MQSeries, do similar configuration steps:

- Create the queue manager by using `crtmqm` from the system console:

```
crtmqm backend.queue.manager
```

- Start the queue manager by using `strmqm` from the system console:

```
strmqm backend.queue.manager
```

- Configure the queue manager by using `runmqsc` from the system console:

```
runmqsc backend.queue.manager
```

- Create the local queue by issuing the following command:

```
define qlocal(BACKEND.QUEUE)
```

- Create the local queue for the transmission queue:

```
define qlocal(BACKEND.TRANSMIT.QUEUE) usage(xmitq)
```

- Create the local definition of the remote queue:

```
define qremote(DEF.OF.BEAN.QUEUE) rname(BEAN.QUEUE) +  
rqmname('bean.queue.manager') xmitq(BACKEND.TRANSMIT.QUEUE)
```

- Create a channel for sending messages using TCP/IP protocol to remote machine `wsnt00` on port `9101`:

```
define channel(BACKEND.CHANNEL) chltype(SDR) trptype(TCP) +  
conname('wsnt00(9101)') xmitq(BACKEND.TRANSMIT.QUEUE)
```

- Create a channel for receiving messages:

```
define channel(BEAN.CHANNEL) chltype(RCVR) trptype(TCP)
```

- Exit from runmqsc session:

```
end
```

After configuring the queue managers, activate listeners on the queue managers, and start the channels. At the Application Server MQSeries:

```
start runmqslsr -t tcp -m bean.queue.manager -p 9101  
start runmqchl -c BEAN.CHANNEL -m bean.queue.manager
```

Similarly, at the back-end MQSeries:

```
start runmqslsr -t tcp -m backend.queue.manager -p 9102  
start runmqchl -c BACKEND.CHANNEL -m backend.queue.manager
```

6.4.2.2 Programming Model

MQSeries for Java implements MQSeries API by using an object-oriented programming model. This approach is different from other MQSeries implementations that uses the procedural programming model. In the MQSeries Java programming interface, an MQSeries object is represented by an object. As an example, a queue is represented by an object of class `MQQueue`. Any call that is related to a queue is implemented using a method in the `MQQueue` class.

MQSeries for Java provides two mechanisms to support Java programs. The MQSeries Client for Java is for client-side implementation. The MQSeries Bindings for Java is for server-side implementation.

MQSeries Client for Java

MQSeries Client for Java is an MQSeries client written in Java language. It follows the Java applet model. The Client for Java enables Java applets to access the MQSeries server. When accessing the applet, the browser will also download the corresponding MQSeries Client for Java classes. Thus, the end-user machine does not need to install any MQSeries component.

The applet opens a server connection channel to MQSeries queue manager via a socket protocol. Unlike ordinary MQSeries channels, this channel is a two-way channel. To connect to a particular MQSeries host, the program should specify the host name, port number and channel name in the MQSeries environment.

MQSeries Bindings for Java

The MQSeries Bindings for Java enables Java applications or servlets in the application server to access MQSeries. This model does not use a channel, but instead, it connects directly to MQSeries queue managers. It uses Java native methods to call MQSeries queue manager API directly. Therefore, this model brings much better performance than the MQSeries Client for Java. You should choose this model for implementing MQSeries interfaces in your WebSphere applications.

The Java component executes in the same machine as the MQSeries server. This means that the MQSeries server should be in the same machine as the application server. A connection to a back-end enterprise MQSeries will be in the server-to-server mode. The application server MQSeries is linked to the back-end enterprise MQSeries.

MQSeries Java Classes

The `ibm.com.mq` package implements MQSeries Client for Java, whereas the `ibm.com.mqbind` package contains MQSeries Bindings for Java classes. Both packages have the same programming interface. Therefore, a Java program can be deployed easily in either environment.

The following are the main classes in the MQSeries for Java:

- *MQQueueManager*. This class represents a connection to a queue manager. Using this connection, the program can access queues in the queue manager by using its `accessQueue` method. The class also provides several utility methods for controlling the connection.
- *MQQueue*. This class represents an MQSeries queue object. It provides methods for inquiry, set, put, and get operations on the queue. The put method puts a message into the queue. The get method retrieves a message from the queue.
- *MQEnvironment*. This class holds MQSeries environment variables. For example, host name, port number and channel name for server connection are contained in this class. It is useful for setting MQSeries Client for Java channel parameters.

Transaction Support

The MQSeries for Java provides some transactional control for queue operations:

- The `MQQueueManager.begin()` signals the queue manager for a new unit of work.
- The `MQQueueManager.commit()` signals a sync point and a commit to the queue manager. The queue manager will make permanent all get or put operations that occurred since the last sync point.
- The `MQQueueManager.backout()` signals a sync point and a rollback to the queue manager. The queue manager will discard all get or put operations that occurred since the last sync point.

6.4.2.3 Application Flow

The basic structure of Java programs that implement MQSeries for Java is as follows:

- Import MQSeries for Java classes. For MQSeries Client for Java, put the following line of the top of your program:

```
import com.ibm.mq.*;
```

For MQSeries Bindings for Java, put:

```
import com.ibm.mqbind.*;
```

- For MQSeries Client for Java, specify host name, channel and port number of the MQSeries Server:

```
MQEnvironment.hostname = "wsnt00.ral.itso.ibm.com";
MQEnvironment.channel = "JAVA.CHANNEL";
MQEnvironment.port = 1414;
```

- Create a connection to a query manager by creating an instance of `MQQueryManager` class:

```
MQQueueManager qMgr;
...
qMgr = new MQQueueManager("wsnt00.queue.manager");
```

- Open a local queue for reading and writing:

```
String localQueueName = "WSNT00.LOCAL.QUEUE";
int openOptions = MQC.MQOO_INPUT_AS_QDEF | MQC.MQOO_OUTPUT;
...
MQQueue qLocal = qMgr.accessQueue(localQueueName, openOptions,
null, null, null);
```

- Open a remote queue for writing:

```
String remoteQueueName = "DEF.OF.REMOTE.QUEUE";
openOptions = MQC.MQOO_OUTPUT;
...
MQQueue qRemote = qMgr.accessQueue(remoteQueueName, openOptions,
null, null, null);
```

- Put a message into a queue:

```
String sendString = "A String from WSNT00...";
MQMessage sendMessage = new MQMessage();
sendMessage.writeUTF(sendString);
// Set put message options as default
MQPutMessageOptions putMessageOptions = new MQPutMessageOptions();
...
qRemote.put(sendMessage, putMessageOptions);
```

- Retrieve a message from a queue:

```

MQMessage receivedMessage = new MQMessage();
// Set get message options: wait up to ten seconds before failed
MQGetMessageOptions getMessageOptions = new MQGetMessageOptions();
    gmo.options = MQC.MQGMO_WAIT;
    gmo.waitInterval = 10000;
...
qLocal.get(receivedMessage, getMessageOptions, MAX_MSG_SIZE);
String receivedString = receivedMessage.readUTF();

```

- Close the queue:

```

qLocal.close();
qRemote.close();

```

- Disconnect from the queue manager:

```

qMgr.disconnect();

```

6.4.3 MQSeries for Java Example

In this section, we describe an example of using MQSeries Java support in the WebSphere environment. The example is deployed as an EJB.

6.4.3.1 The CardServices Inc.

The CardServices Inc. is providing credit card services to other companies. One of its services is to provide credit card statuses and ratings as an online service. Our XtremeTravel online site can use this service to verify customers' credit cards when they reserve or arrange travel packages.

The service is delivered through a CardInfoEJB. The EJB makes use of a utility class MQConn that handles all operations with MQSeries. It uses MQSeries Bindings for Java for better performance. The EJB runs on the WebSphere Application Server.

For simulating the CardServices legacy system, we create a Java application, CardHost.java. The connection from application server machine and simulated CardServices host uses the configuration as we discussed in 6.4.2.1, "Setting Up MQSeries for Java" on page 365.

6.4.3.2 Example: MQConn Utility Class

The following example is a Java class that implements the MQSeries connection. The MQConn provides basic MQSeries services. The class is a server-side implementation of MQSeries by using the com.ibm.mqbind package. A similar class can be developed for client-side implementation by changing the package to com.ibm.mq and setting host name, channel and port number fields into a MQEnvironment static class. An EJB can use the class for accessing enterprise applications via MQSeries.

```

package com.ibm.redbook.sg245460;

import java.io.*;
import java.util.*;
import com.ibm.mqbind.*;          // For MQSeries Bindings for Java

public class MQConn {

    private String queueManagerName = "mq.queue.manager";// Query Manager to connect
    private String localQueueName = "mq.local.queue";      // Queue name
    private String remoteQueueName = "mq.remote.queue";    // Queue name
    private MQQueueManager qMgr;// define a queue manager object
    private MQQueue      localQueue;
    private MQQueue      remoteQueue;

    private final static int MAX_MSG_SIZE=256;

    public int open() {
        int rc=0;
        int openOptions = 0;
        try {
            // Creating connection to queue manager
            qMgr = new MQQueueManager(queueManagerName);
            // Accessing local queue
            openOptions = MQC.MQOO_INPUT_AS_Q_DEF | MQC.MQOO_OUTPUT ;
            localQueue = qMgr.accessQueue(localQueueName,openOptions,null,null,null);

            if( remoteQueueName.equals(localQueueName) )
                remoteQueue = localQueue;
            else {
                // Accessing remote queue
                openOptions = MQC.MQOO_OUTPUT;
                remoteQueue = qMgr.accessQueue(remoteQueueName,openOptions,null,null,null);
            }
        } catch (MQException e) {
            System.out.println("MQException: " + e.completionCode + ":" + e.reasonCode);
            rc = e.reasonCode;
        }
        return rc;
    } // end of open()

    public int open(String queueManagerName, String localQueueName, String remoteQueueName) {
        this.queueManagerName = queueManagerName;
        this.localQueueName = localQueueName;
        this.remoteQueueName = remoteQueueName;
        return open();
    }

    public int open(String propertyFileName) {
        try {
            Properties p = new Properties();
            p.load(new FileInputStream(propertyFileName));
            queueManagerName = p.getProperty("mq.queue.manager");
            localQueueName = p.getProperty("mq.local.queue");
            remoteQueueName = p.getProperty("mq.remote.queue");
        } catch (IOException e) {
            System.out.println("IOException: " + e.getMessage());
        }
        return open();
    }
}

```

Figure 354. MQConn Class (1/2)


```

public void close() {
    try {
        localQueue.close();
        remoteQueue.close();
        qMgr.disconnect();
    }
    catch (MQException e) {
        System.out.println("MQException: " + e.completionCode + ":" + e.reasonCode);
    }
} // end of close()

public int send(String message) {
    int rc=0;
    try {
        // Create an MQ message, fill it with the message
        MQMessage msg = new MQMessage();
        msg.writeUTF(message);
        // Specify put message options
        MQPutMessageOptions pmo = new MQPutMessageOptions();

        // Put the message on the queue
        remoteQueue.put(msg,pmo);
    }
    catch (MQException e) {
        System.out.println("MQExcpetion: " + e.completionCode + ":" + e.reasonCode);
        rc = -e.reason.code;
    }
    catch (java.io.IOException e) {
        System.out.println("IOException: " + e.getMessage());
    }
    return rc;
} // end of send()

public int receive(StringBuffer message) {
    int rc=0;
    try {
        MQMessage msg = new MQMessage();
        // Specify get message options
        MQGetMessageOptions gmo = new MQGetMessageOptions();
        gmo.options = MQC.MQGMO_WAIT;
        gmo.waitInterval = 10000; // Wait 10 seconds before it failed

        // Getting the message from the queue
        localQueue.get(msg,gmo,MAX_MSG_SIZE);
        message.setLength(0);
        message.append(msg.readUTF());
    }
    catch (MQException e) {
        rc = e.reasonCode;
        if(rc != MQException.MQRC_NO_MSG_AVAILABLE) {
            System.out.println("MQException: " + e.completionCode + ":" + e.reasonCode);
            rc = -rc;
        }
    }
    catch (java.io.IOException e) {
        System.out.println("IOException: " + e.getMessage());
    }
    return rc;
} // endof receive()
} // end of MQConn

```

Figure 355. MQConn Class (2/2)

6.4.3.3 Example: CardInfoEJB

The CardInfoEJB queries credit card status and ratings from CreditServices Host. To deploy this EJB, follow the instructions in 4.2.3, “Deploying an EJB” on page 192. The EJB can be called by an applet, a servlet or other Java beans using the EJB calling mechanism (see 4.3, “Coding WebSphere EJB Clients” on page 207).

The CardInfoEJB uses the MQConn class as described in 6.4.3.2, “Example: MQConn Utility Class” on page 371.

To set MQSeries object names, we use a property file that is put in <ASROOT>/properties/mqconn.properties, with the following contents:

```
# WebSphere Application Server
# IBM Redbook SG245460
#
# Sample property file for MQSeries Connection
#
#Wed May 17 11:02:12 EDT 1999

mq.queue.manager = bean.queue.manager
mq.local.queue = BEAN.QUEUE
mq.remote.queue = DEF.OF.BACKEND.QUEUE
```

Figure 356. MQSeries Connection Properties in mqconn.properties File

Figure 357 shows the bean implementation for CardInfoEJB:

```

package com.ibm.redbook.sg245460.card;

import java.rmi.RemoteException;
import java.security.Identity;
import java.util.Properties;
import javax.ejb.*;
import java.lang.*;
import com.ibm.redbook.sg245460.*;

public class CardInfoEJBBean implements javax.ejb.SessionBean {
    private javax.ejb.SessionContext mySessionCtx = null;

    public CardInfo getCardInfo(String cardNo) {

        int rc;

        MQConn mqc = new MQConn();

        mqc.open("properties/mqconn.properties");

        // Send Card Number to Back End Host
        mqc.send(cardNo);

        // Wait and receive the response;
        StringBuffer respbuff = new StringBuffer(256);
        mqc.receive(respbuff);

        // Parse the response and build the response object
        CardInfo ci = new CardInfo();
        String response = new String(respbuff);
        ci.setCardNo(response.substring(0,16));
        ci.setCardName(response.substring(17,19));
        ci.setHolderName(response.substring(20,36));
        ci.setCardStatus(Integer.valueOf(response.substring(37,39)).intValue());
        ci.setCardRating(Integer.valueOf(response.substring(40,42)).intValue());
        ci.setStatusDesc(response.substring(43,59));

        mqc.close();

        return ci;
    }

    public void ejbActivate() throws java.rmi.RemoteException {
    }
    public void ejbCreate() {
    }
    public void ejbPassivate() throws java.rmi.RemoteException {
    }
    public void ejbRemove() throws java.rmi.RemoteException {
    }
    public void setSessionContext(javax.ejb.SessionContext ctx) throws java.rmi.RemoteException {
        mySessionCtx = ctx;
    }
}

```

Figure 357. MQSeries Example: CardInfoEJBBean

6.5 Using TXSeries for WebSphere Application

IBM TXSeries is a transactional middleware solution for UNIX and Windows NT platforms. The TXSeries provides:

- An advanced transaction processing server that combines the technology from Customer Information Control System (CICS) and Transarc's Encina transaction processing products.
- Connectivity suite for e-business such as:
 - CICS Client and Encina Client
 - CICS Gateway for Java
 - CICS Internet Gateway

- CICS Gateway for Lotus Notes

You can find more information about IBM TXSeries and other IBM Transaction System products at <http://www.software.ibm.com/ts>.

In relation to WebSphere applications, the TXSeries can provide:

- Back-end transaction processing system
- Connectivity to IBM Transaction Servers such as the well-known CICS

Since WebSphere applications use Java as their underlying platform, the CICS GATEWAY for Java, which is in the TXSeries package, will be the only connectivity tool to access Transaction Servers.

IBM also has several products that bring connectivity between Java programs and Transaction Servers:

- CICS Connector, which is a set of Java classes for developing Java clients that can access CICS servers
- CICS Transaction Gateway, which includes CICS GATEWAY for Java and CICS Universal Clients in one product

These two products provide similar functions to CICS GATEWAY for Java.

Since we are discussing TXSeries, we will focus on CICS GATEWAY for Java. Techniques discussed in this section should apply for all products mentioned above.

6.5.1 IBM CICS Gateway for Java

The CICS GATEWAY for Java is a set of Java applications and class libraries that enables Java programs to access CICS servers or TXSeries servers. It provides secure access and a set of development class libraries. The product is available as a separate package or as a part of IBM CICS Transaction Gateway.

The CICS GATEWAY for Java requires CICS Universal Clients or CICS Client to access CICS servers. The CICS Universal Clients are available in IBM CICS Transaction Gateway.

The CICS Gateway for Java has a server-side Java application that acts as the gateway to the CICS Client. Any Java program (which can be a Java application, applet, servlet, or EJB) communicates with the gateway by using a proprietary socket-based protocol, or through HTTP, HTTPS or SSL protocols.

The CICS Client acts as the interface to CICS servers. It can communicate with multiple CICS servers with various protocols, such as APPC and TCP/IP. It handles communication complexities with CICS servers and routes client requests to the intended CICS server.

A Java program can issue a CICS ECI or EPI request to the gateway. The gateway translates the request into ECI or EPI calls and gives it to the CICS Client, then passes them to the intended CICS server. After processing the request, the CICS server returns the result back to the Java program via the CICS client and the gateway.

Figure 358 on page 377 summarizes various alternatives for accessing CICS servers from a Java program. For security reasons, the gateway is usually in the same machine as the application server. A servlet or EJB accesses the gateway using a local TCP/IP connection. The servlet is for accessing CICS applications from a client browser. The servlet can access the gateway directly or use EJBs. The EJB can encapsulate CICS transactions to represent a particular business object. In distributed component architecture, EJBs in other machines should communicate with application server EJBs to access the CICS server.

Other Java applets or applications outside the application server can also access the gateway. In this section, we focus on access from the Application Server environment.

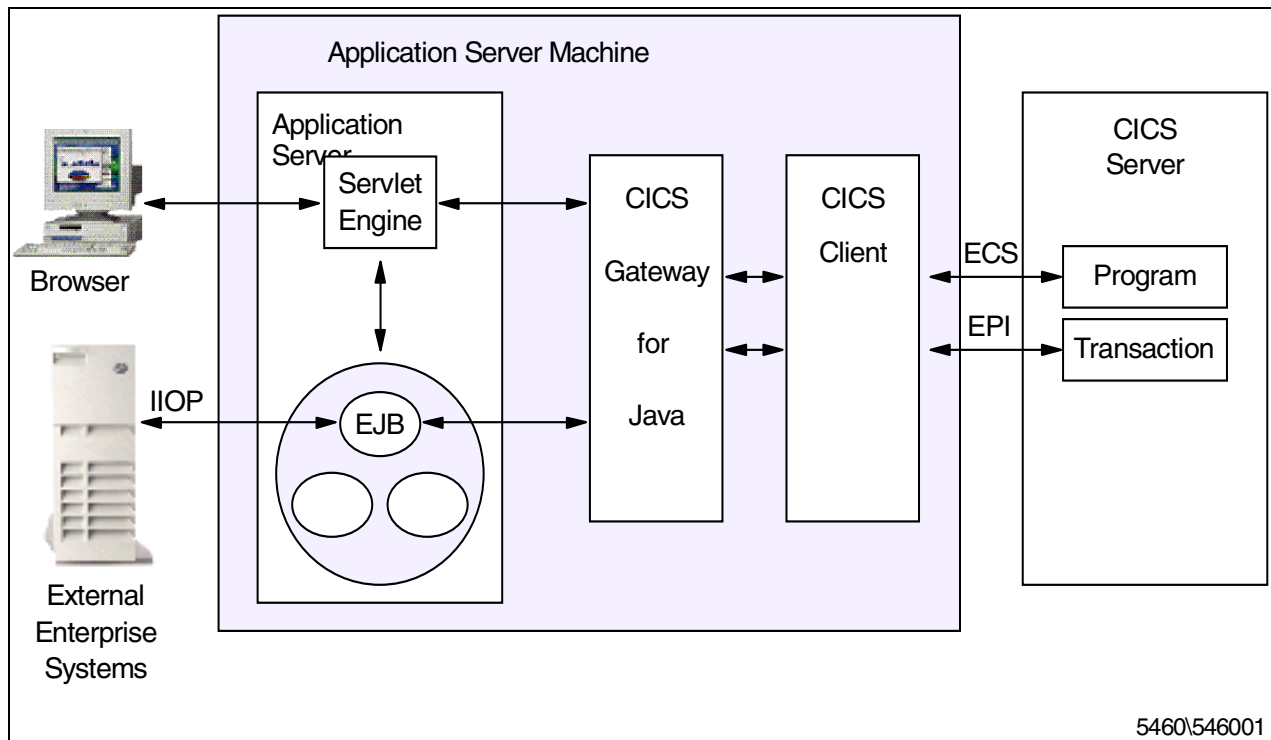


Figure 358. CICS Gateway for Java Architecture

6.5.1.1 Programming Model

The CICS Gateway for Java provides a set of classes for developing Java clients. The classes are put in several packages:

- `ibm.cics.jgate.client` contains classes for creating connections and requests.
- `ibm.cics.jgate.security` contains security support classes.
- `ibm.cics.jgate.epi` contains EPI support classes.

The Gateway Server

The gateway server is implemented as a server-side Java application `ibm.cics.jgate.server.JGate`. In CICS Gateway for Java, any Java program acts as a client of JGate. Then, the JGate translates requests into ECI or EPI calls and gives them to the CICS Client. To access the gateway servers, the Java program creates a connection to JGate.

If the Java program and the CICS Client are in the same machine, you may bypass the gateway server. In this case the Java program connects to the CICS Client directly using a local protocol. You don't need to run the gateway server.

Gateway Connection

An instance of the `ibm.cics.jgate.client.JavaGateway` class represents a logical connection from a Java program to the gateway server. A Java program may create logical connections to several gateway servers by creating several `JavaGateway` objects.

To locate a gateway server, specify a URL connection parameter. A local connection (Java Client and gateway server are in the same machine) uses the following URL:

```
local://
```

A remote connection URL syntax is:

```
<protocol>://<gateway_server_address>:<port>/
```

where `<protocol>` is `http` or `tcp`; `<gateway_server_address>` is the fully qualified host name or the IP address of the gateway server; and `<port>` is the TCP port on which the gateway server is listening. The default protocol is `tcp`. The HTTP protocol is preferred when the network uses an HTTP proxy firewall.

In WebSphere environments, normally the application server and the CICS Client are in the same machine. Therefore, we recommend using a local protocol for secure and better performance.

Interface Model

Any non-CICS application that accesses a CICS server requires an external interface into the CICS system. CICS provides two ways for creating external interfaces, ECI (External Call Interface) and EPI (External Presentation Interface). These interface models define the construct of requests and responses.

- In the ECI model, the client calls a CICS program in a CICS server. The call format is application defined, which generally consists of program name and program parameters. The client uses CICS COMMAREA to represent both the request and the response to and from the CICS server. The client sends the request in COMMAREA. The CICS server returns the responses and return codes in COMMAREA.
- In the EPI model, the client acts as a 3270 terminal of a CICS server. The client sends 3270 data streams to the CICS server to initiate a CICS transaction or to update CICS resources. The server responds with 3270 data streams.

A Java client sends CICS requests as `GatewayRequest` objects. Based on the external interface model, there are two forms of requests:

- As an object of `ibm.cics.jgate.client.ECIRequest` class by using the ECI model. The object contains a COMMAREA structure for both request and CICS server response. The object is also used for CICS server responses.
- As an object of `ibm.cics.jgate.cllient.EPIRequest` class by using the EPI model. The object contains 3270 data streams for both request and CICS server response. The object is also used as a CICS server response.

Both ECIRequest and EPIRequest extend `ibm.cics.jgate.client.GatewayRequest` class.

JavaGateway Methods

The `JavaGateway` is the class for connection to the Gateway server. It provides several constructor methods for performing operations on the connection:

- `JavaGateway(String urlString)` creates a connection as specified by connection URL `urlString`.
- `setURL(String urlString)` sets a connection URL to `urlString`.
- `open()` opens the connection.
- `flow(GatewayRequest)` sends an `ECIRequest` or `EPIRequest` object. The method will wait until the CICS server finishes processing. The CICS server response is returned in the same `ECIRequest` or `EPIRequest` object.
- `close()` closes the connection.

6.5.1.2 Application Flow

The general structure of the CICS Gateway for Java programs is:

- Import CICS Gateway for Java classes. Insert the following lines at the top of your Java program:

```
import ibm.cics.jgate.client.*;
```

- Create a connection to the Gateway server by creating an instance of `JavaGateway` class and specifying the connection's URL. For example, we use a local connection:

```
urlString connUrl = "local:";
...
JavaGateway conn = new JavaGateway();
conn.setURL(connURL);
...

```

As an alternative, you may remove the `setURL` statement and use another `JavaGateway` constructor with a URL parameter:

```
JavaGateway conn = new JavaGateway(connURL);
```

- Open the connection:

```
conn.open();
```

- If you use the ECI model, create an ECI request object and fill its parameters:

```
String cicsServerName = "CICS001";
String cicsUserId = "USRDEV01";
String cicsPassword = "tryagain";
String cicsProgram = "ORDFOOD1";
byte[] cicsCommarea = String("01SANDWICH02DIETCOKE").getBytes();
...
ECIRequest eciRequest = new ECIRequest(cicsServerName,
    cicsUserId,
    cicsPassword,
    cicsProgram,
    cicsCommarea,
    ECIRequest.ECI_NO_EXTEND,
    ECIRequest.ECI_LUW_NEW);
```

Send the request and wait for the CICS server response:

```
conn.flow(eciRequest);
```

Retrieve server results in the COMMAREA, check the return code and the abend code, if any:

```
String serverResponse = new String(eciRequest.Commarea);
int cicsRc = eciRequest.Cics_Rc;
String abendCode = eciRequest.Aabend_Code;
```

- If you use the EPI model, create an EPI request object. Here, the process is somewhat complicated, since in the EPI model, the client acts as a terminal. We should make sure that the client follows the proper presentation sequence:

```
String sysName;
String netName;
String devType;
EPIRequest epiRequest = null;
String tranId;
String cmdData;
int cicsRc;
String abendCode;
...
// Add a pseudo terminal to system sysName
epiRequest = EPIRequest.addTerminal(sysName,netName,devType);
conn.flow(epiRequest);
...
// Start Transaction
// For example, we will call CICS command level interpreter CECI.
tranId = "CECI";
// Send escape (0x27), then at row 1 (0x20) col 1 (0x20) put CECI
cmdData = "\x27 CECI"; // 0x27 0x20 0x20 'C' 'E' 'C' 'I'
epiRequest.startTran(tranId,cmdData.getBytes(),cmdData.length);
conn.flow(epiRequest);
...
// Loop, check for events,
// If the event is Converse, you can send a reply
// and wait until END_TRAN event.
while(true) {
    ...
    // Wait for event ...
    epiRequest.getEvent(EPIRequest.EPI_WAIT,1000);
    conn.flow(epiRequest);

    // Send command, only if the system is ready
    // For example, send "F3" (0x33) 'Exit' to CECI
    cmdData = "3 "; // 0x33 0x20 0x20
    if(epiRequest.event == EPIRequest.EPI_EVENT_CONVERSE) {
        epiRequest.sendReply(cmdData.getBytes(),cmdData.length);
        conn.flow(epiRequest);
    }

    if(epiRequest.event == EPIRequest.EPI_EVENT_END_TRAN) {
        cicsRc = eciRequest.Cics_Rc;
        abendCode = eciRequest.Aabend_Code;
        break; // exit from loop
    }
    ...
}
...
```


- After processing has finished, close the connection to release resources:

```
conn.close();
```

6.5.2 Setting Up CICS Gateway for Java for WebSphere

The requirement for using CICS Gateway for Java support in the WebSphere environment is to set up Java path information in the WebSphere Java Engine setup.

6.5.2.1 Setting Up System CLASSPATH Variable

Setting Java library paths in system environment variables may be useful for developing and debugging purposes. Add the following path into the CLASSPATH variable:

```
<JGATE_HOME>/classes
```

where <JGATE_HOME> is your CICS Gateway for Java installation directory.

Add the following path to the library path variable:

```
<JGATE_HOME>/bin/<platform>
```

where <platform> can be "nt", "aix", "mvs", "os2" or "solaris".

6.5.2.2 Setting Up WebSphere CLASSPATH and User LibPath

In the WebSphere Administration, you should also include Java paths into the application server classpath and user libpath.

- Log on to the WebSphere Administration Tool using the administrator user ID.
- Select **Setup > Java Engine** to bring up the Java Engine setup form.
- On the Path tab add the following application server classpath entry field:

```
<JGATE_HOME>/classes
```

- Add <JGATE_HOME>/bin/<platform> to the user libpath entry field.

After changing this path information, restart both the Web server and WebSphere. In AIX, you may have to kill the WebSphere processes before restarting the Web server.

6.5.2.3 Configuring CICS Server Listener and CICS Client

CICS Clients access the CICS server by creating a connection to the CICS server listener service. The link can use various protocols, such as TCP/IP and APPC.

To configure a listener service in the CICS server using the TCP/IP protocol:

- Add a service entry into the TCP/IP service file on the CICS server machine. In Windows NT, the file is \winnt\system32\drivers\etc\services. In AIX, the file is /etc/services. Add the following line into the file:

```
<service_name> <port>
```

- Use an existing CICS region, or add a new region by using:

```
cicscp -v create dce -R
cicscp -v create region <region_name>
```

- Add a listener definition into the CICS server region:

```
cicsadd -c ld -r <region_name> -p <listener_name> Protocol=TCP
```

```
TCPService=<service_name>
```

where <listener_name> is any name for identifying the listener.

For example, if we create a new region CICSNT01, and set a listener CICSLS01 using CICSTCP01 as <service_name> and 1234 as <port>:

- Add this line into the TCP/IP service file:

```
CICSTCP01    1234 # CICS Listener
```

- Execute the following line in the command line:

```
cicscp -v create dce -R
cicscp -v create region CICSNT01
cicsadd -c ld -r CICSNT01 -P CICSLS01 Protocol=TCP TCPService=CICSTCP01
```

- Start the region by executing:

```
cicscp -v start region CICSNT01
```

To configure the CICS client to access this listener:

- Add into the CICS client configuration file, CICSCLI.INI, the following lines:

```
Server = <server_name>
Description = TCP/IP Server
Protocol = TCPIP
NetName = <server_host_name>
Port = <port>
```

where <server_name> is an arbitrary name that uniquely identifies the CICS server.

- Start a connection to the CICS server:

```
cicscli /S=<server_name>
```

For example, if the previous CICS server example is in the host wsnt00.itso.ral.ibm.com, add these lines into CICSCLI.INI:

```
Server = CICSSV01
Description = Connection to TCP/IP Server Example
Protocol = TCPIP
NetName = wsnt00.itso.ral.ibm.com
Port = 1234
```

Start the CICS client connection to the CICS server by using:

```
cicscli /S=CICSSV01
```

Verify that the connection is working:

```
cicscli /l
```

6.5.2.4 Setting Up and Running the Gateway

You need to set up and run the Java Gateway application only if you use Network JavaGateway objects to connect to the CICS Client in a remote machine. If the Java program and the CICS Client are in the same machine, you do not need to run the JavaGateway application and use the local protocol. In this case, the JavaGateway object in the Java program communicates directly to the CICS Client.

The Java Gateway application has a property file, Gateway.properties. You need to set several parameters to connect the Gateway applications to the CICS Client.

- Open <JGATE_HOME>/bin/Gateway.properties file
- Specify the network protocol handler and its parameters:

```
protocol@<protocol_name>.handler = <protocol_handler_name>  
protocol@<protocol_name>.parameters = <parm1=value1;parm2=value2 ...>
```

For example, if the JavaGateway application is listening on port 9090:

```
protocol@tcp.handler=com.ibm.cics.jgate.server.TCPHandler  
protocol@tcp.parameters= port=9090
```

This will correspond to connection URL tcp://<gateway_host_name>:9090/ for the JavaGateway object in the Java program.

To start the gateway, enter the following command in the command line:

```
JGate
```

6.5.3 CICS Gateway for Java Example

In this section, we give an example using CICS Gateway for Java in the WebSphere environment.

6.5.3.1 Example: CICSAccess Utility Class

The following example is a Java utility class that provides methods to access the CICS server by using CICS Gateway for Java. The class uses the ECI model for creating CICS requests.

To set CICS connection parameters, the class reads a property file in <ASROOT>/ properties/cicsconn.properties. For example, the file might look like this:

```
# WebSphere Application Server  
# IBM Redbook SG245460  
#  
# Sample property file for CICS Connection  
#  
#Wed May 17 11:02:12 EDT 1999  
  
cics.url = local:  
cics.user = CICSUSER  
cics.password = LETMEIN  
cics.server = CICS01
```

Figure 359. CICS Connection Properties in cicsconn.properties File

Figure 360 and Figure 361 show the code for the CICSAccess class:

```

package com.ibm.redbook.sg245460;

import java.io.*;
import java.util.*;
import ibm.cics.jgate.client.*;

public class CICSAccess {

    private String urlString = "local:";
    private String cicsUserId = "CICSUSER";
    private String cicsPassword = "";
    private String cicsServerName = "";
    private String cicsProgram = "";
    private byte[] cicsCommarea = "";
    private int    cicsCommareaLength = 0;

    private JavaGateway cicsConn = null;
    private ECIRequest eciRequest = null;

    public void open() {
        try {
            cicsConn = new JavaGateway(urlString);
            cicsConn.open();
        } catch (IOException) {
            e.printStackTrace();
        }
    }

    public int open(String propertyFilename) {
        try {
            Properties p = new Properties();
            p.load(new FileInputStream("properties/cicsconn.properties"));

            urlString      = p.getProperty("cics.url");
            cicsUserId     = p.getProperty("cics.user");
            cicsPassword   = p.getProperty("cics.password");
            cicsServerName = p.getProperty("cics.server");

        } catch (IOException e) {
            e.printStackTrace();
        }
        open();
    }

    public int sendRequest(String cicsProgram, String cicsCommarea, int cicsCommareaLength) {
        int rc=0;
        try {
            byte abCommarea = new byte[ cicsCommareaLength ];
            System.arraycopy( cicsCommarea.getBytes(), 0, abCommarea, 0,
                Math.min(abCommarea.length, cicsCommarea.length()));
        }
    }
}

```

Figure 360. CICSAccess Class (1/2)

```
public String getResponse() {  
    if( eciRequest.Cics_Rc = 0 ) {  
        return new String(eciRequest.Commarea);  
    } else {  
        return "ABEND:" + eciRequest.Abend_Code);  
    }  
}  
  
public void close() {  
    try {  
        jgaConection.close();  
    } catch(IOException e) {  
        e.printStackTrace();  
    }  
}
```

Figure 361. CICSAccess Class (2/2)

Chapter 7. WAS 3.0, Site Analyzer Technology Preview

This chapter provides a technology preview of a WebSphere product function that will be available with WebSphere Application Server V3.0. It is currently available in beta only. Some of the screens and functions will change before general availability. The purpose of this chapter is to give you exposure to the product and its functions.

As a Web site becomes more complex, the job of managing the site becomes more critical. The Web site contents, links and user activities, and the tasks for managing these resources, can grow too fast and too complicated to be managed manually. Some examples of tasks that represent crucial site operations are:

- Checking that the content is the correct version
- Checking that the links to the contents and to other supporting sites are working with sufficient bandwidth
- Monitoring the usage of particular content or a link

These tasks, which are normally what webmasters do, are becoming important factors for building a successful Web site.

The WebSphere Site Analyzer is aimed at solving these problems. The Site Analyzer is a set of software programs and tools used to manage Web site operations. It provides:

- Web content analysis
- Usage analysis for monitoring resource usages
- Statistics using user-defined reports

This chapter focuses on using the WebSphere Site Analyzer to manage Web Site operations. For information about software requirements, installation and setup for the WebSphere Site Analyzer, see the following section.

7.1 Installing WebSphere Site Analyzer

The Site Analyzer is a tool for analyzing Web sites. It can check Web site contents, analyze the usage of the content, and produce reports. In this section, we work with the installation of Site Analyzer.

The Site Analyzer is a client/server application. The Site Analyzer server does not have to run in the same machine as the Web Server and you should consider running it on a separate machine, because the Site Analyzer server uses lots of CPU processing time to perform its analysis. The client machine should have the Site Analyzer client component. It can be in the same machine as the Site Analyzer server.

The Site Analyzer uses IBM DB2 UDB V5.2 for its databases.

Before installing the Site Analyzer, there are several steps that need to be done:

- As a prerequisite, you should have installed JDK 1.1.7b (on Windows NT and Solaris) or JDK 1.1.6 with fix pack 8 on AIX, a database such as DB2 UDB Workgroup or Enterprise Edition, and a Web browser such as Netscape

Communicator 4.x or Internet Explorer 4.x. The database can be in the local machine or in a remote machine.

Note: If you don't want to install the JDK, then Site Analyzer will automatically install the appropriate JRE for you.

- Create or use an existing database for Site Analyzer purposes. If you use a remote database, you should configure your local machine for a remote interface. When using DB2, you can use the Client Configuration Assistant, as described in 2.1.7.3, "Configuring Remote Interface" on page 55.
- At this point, you are ready to install the Site Analyzer.

The installation package provides an installation program. In Windows NT, it is a standard Windows installation using `setup.exe`. The installation program consists of a sequence of dialogs as shown in Figure 362 on page 388 to Figure 365 on page 390.

- After the first Welcome dialog box, enter a destination directory.
- The next dialog box asks you to select components. You can install both server and client components in the same machine. In the client machine, you should install only the client component.
- The subsequent dialog boxes set the program folder name, copy the files into the destination directory, and finish the installation process.

Note: Site Analyzer packages and silently installs DB2 UDB and it can include the DB2 UDB national language group1 support.



Figure 362. Site Analyzer Installation Welcome Dialog

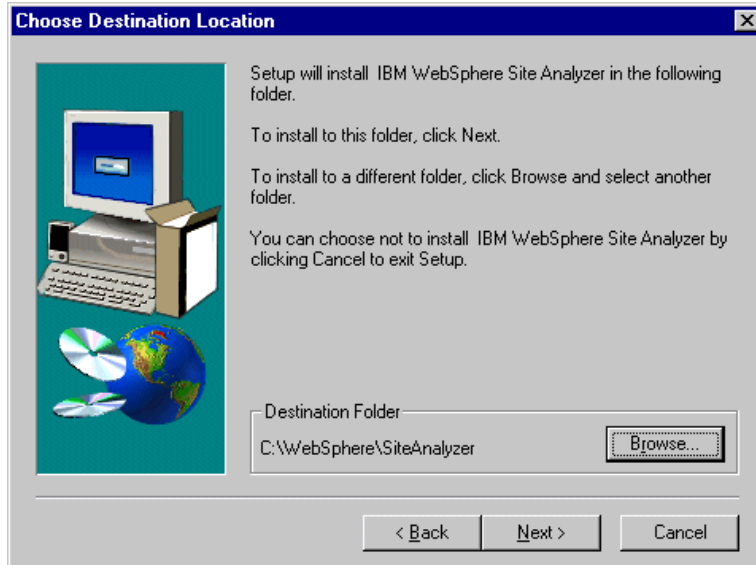


Figure 363. Site Analyzer Installation - Specifying Destination Directory

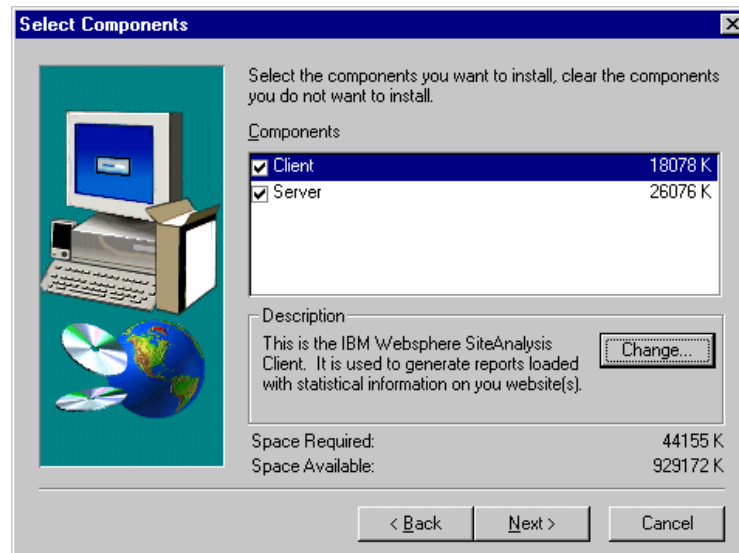


Figure 364. Site Analyzer Installation - Select Components

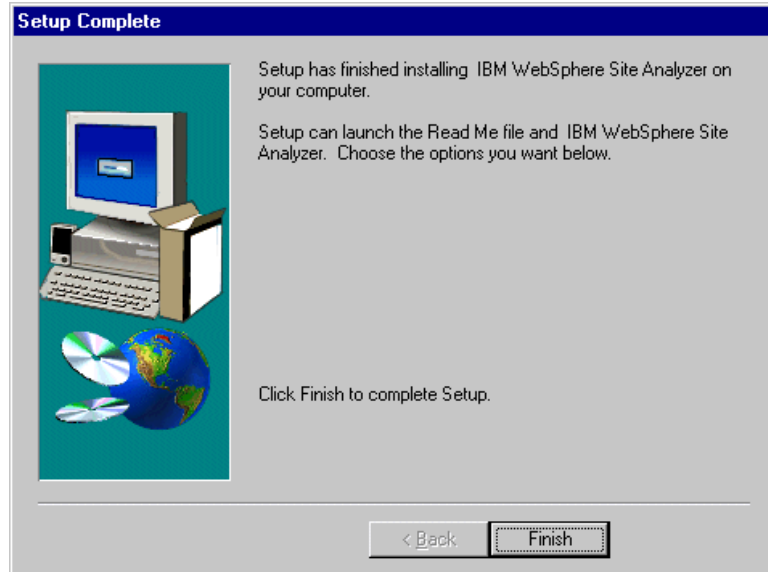


Figure 365. Site Analyzer Installation - Last Dialog Box

7.1.1 First Time Setup

In our Site Analyzer installation process, we did not configure the Site Analyzer to use the database that we had created before. When you use the Site Analyzer for the first time, it will automatically invoke a Startup Wizard (Figure 366 on page 391) to configure database connections and server addresses. To invoke the Site Analyzer, click **Start -> Programs IBM WebSphere -> Site Analyzer 3.0 -> Site Analyzer**.

The Startup Wizard consists of a sequence of tab dialog boxes. At this point, only the Server tab and Database tab are important.

- In the Server tab (Figure 367 on page 391), specify the TCP/IP host name and port number for the Site Analyzer server. If you have not installed a Site Analyzer server, you won't be able to test the connection. The test button is applicable for the client only while the server is running. The test allows the user to see if the client can in fact connect to the server with the specified configuration.
- In the Database tab (Figure 368 on page 392), specify the database connection parameters, which include the connection URL, database user ID and password, and the JDBC driver. To make sure that the connection works, click the **Test Connection** button.

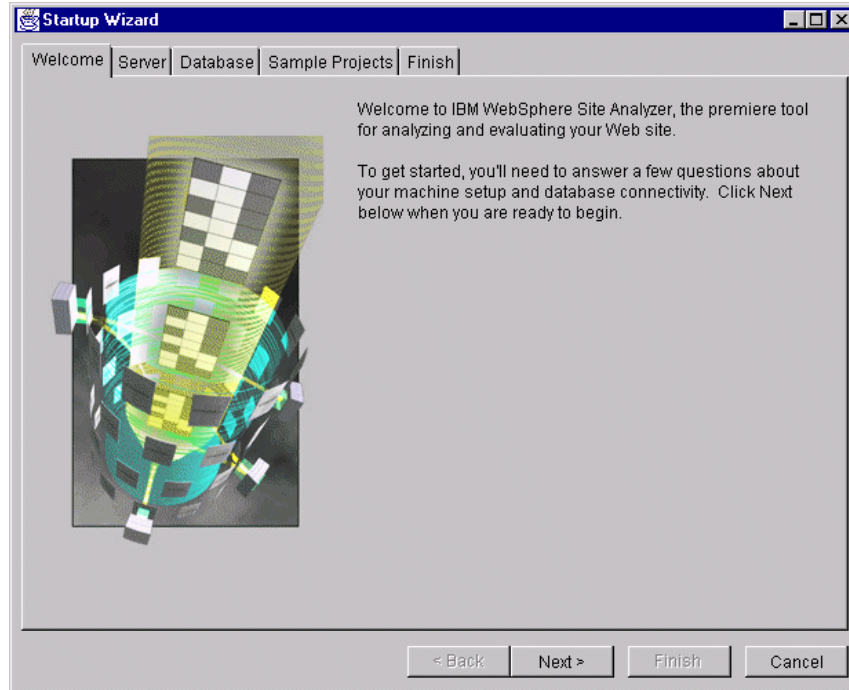


Figure 366. Site Analyzer Startup Wizard for First Time Setup

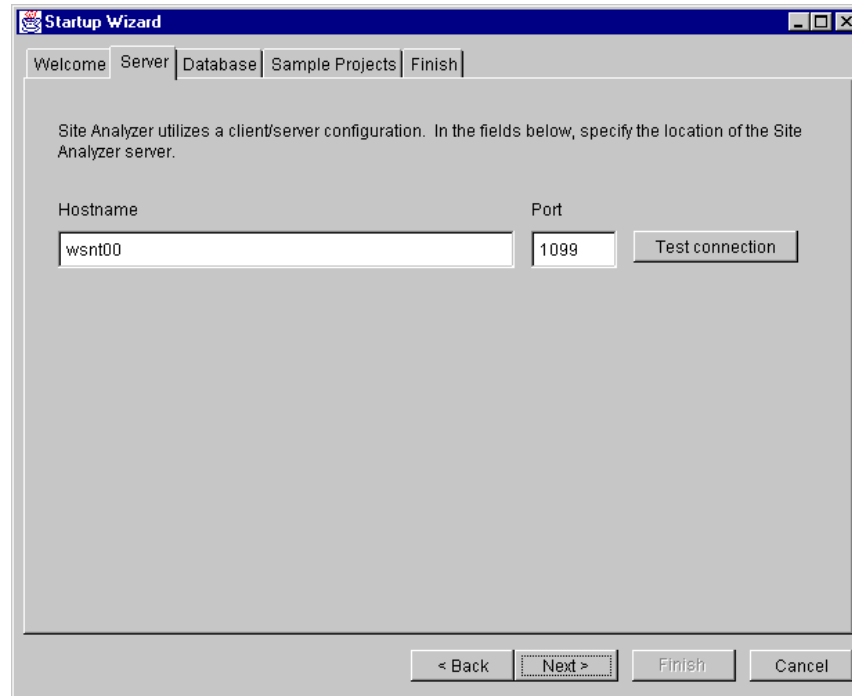


Figure 367. Site Analyzer Startup Wizard - Specifying Site Analyzer Server Parameter

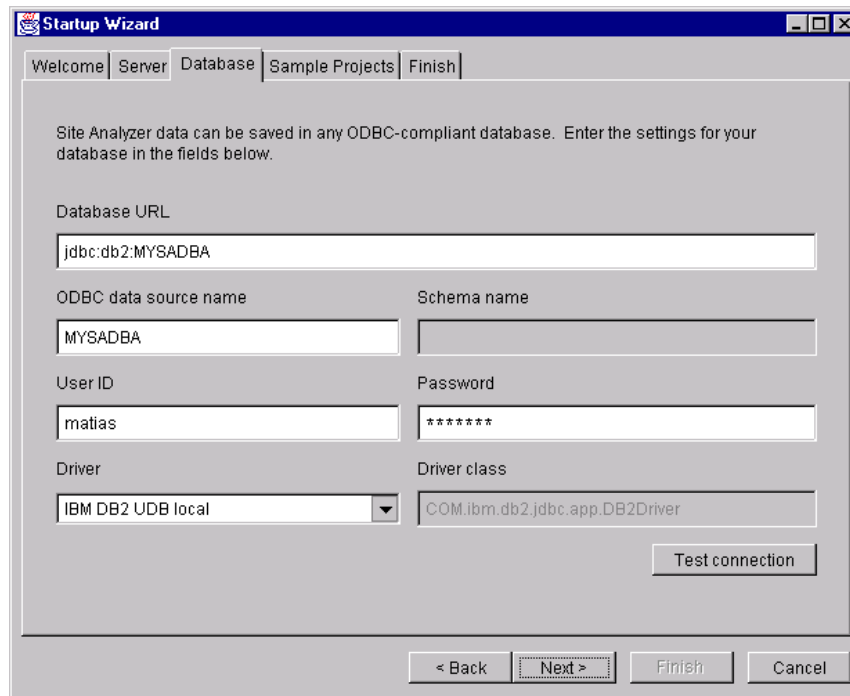


Figure 368. Site Analyzer Startup Wizard - Specifying Database Connection

After you click **Finish** in the last Startup Wizard Tab dialog box, the wizard will automatically start the Site Analyzer.

7.2 A First Look at the Site Analyzer

This section gives a brief tour of Site Analyzer. After you have installed and set up the Site Analyzer, you can run it by clicking **Start -> Program -> IBM WebSphere -> Site Analyzer V3.0**. This will bring up Site Analyzer Project Center (Figure 369), which in fact is a Site Analyzer client application. The Project Center provides several navigation mechanisms:

- Menu bar.
- Tool bar. Some tools in the tool bar are contextual. They will operate on a selected item on the Project tree or on the right-frame list.
- Project tree in the left frame. Each tree node has a contextual list in tabular format in the right frame. When you select a tree node, the right frame will display a table containing items or information of the node type.
- Contextual menu that is invoked when you right-click any object in the Project tree or right-frame list.

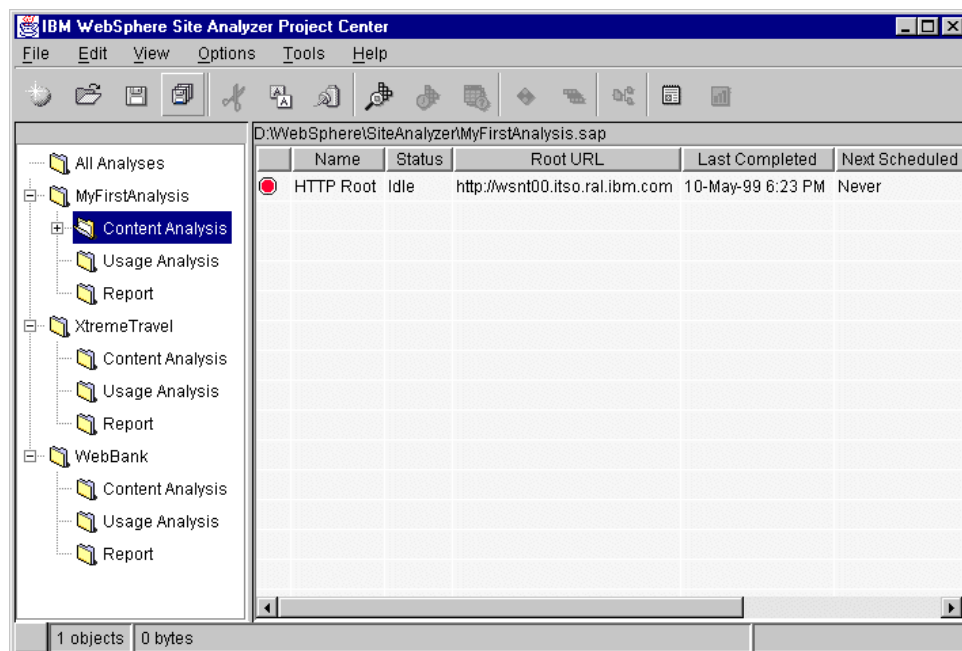


Figure 369. WebSphere Site Analyzer Project Center

7.2.1 Site Analyzer Users

The Site Analyzer has two user roles:

- Administrator, which can perform every available action including configuring Site Analyzer, creating analysis, running analysis and viewing the results
- Client, which can view analysis results only

Table 34 summarizes actions that Site Analyzer user roles can perform. An "X" under Administrator or Client column specifies that the user in that role is allowed to do the actions:

Table 34. User Authorization

Action	Administrator	Client
Create, edit and delete projects	X	X
Create, edit and delete Content/Usage Analysis	X	
Run and Schedule Content/Usage Analysis	X	
Site Surveyor	X	X
Quick Find	X	X
Create, edit and delete Analysis Report Element	X	X
Create report	X	X

The Site Analyzer user setup follows operating system user setup. However, user privilege is defined and verified in the database. The Site Analyzer does not

provide its own user logon and setup tool. The database administrator grants the corresponding privileges to the user.

7.2.2 Site Analyzer Analysis and Projects

In Site Analyzer, the analysis is a set of definitions for a particular site management task. The definition includes:

- Analysis name and type
- Target site information
- Analysis parameters

The Site Analyzer has two forms of analysis:

- Content analysis
- Usage analysis

The Site Analyzer organizes analysis tasks into Site Analyzer projects. Each project defines a set of content analysis tasks, a set of usage analysis tasks and a set of reports.

The Site Analyzer stores project information into a file, with a .sap extension. The project file resides in the client machine and contains links to various items in the database.

7.2.3 Site Analyzer Architecture

The Site Analyzer is a Java client/server application. The Site Analyzer server runs as a background process in a server machine. The Site Analyzer client is a Java Swing user interface application. You can install the client component in the same machine as the server. The connection between the client and the server uses socket protocol on a specified port. The server host name and service port number are set in Startup Wizard, which is invoked during the client's first-time run.

The Site Analyzer stores its data in a DB2 UDB database. The Site Analyzer server and the database engine may be in different machines. In this case, you can create either a remote database connection between them or use JDBC net drivers.

The Site Analyzer performs the analysis by two main mechanisms:

1. Using a Web robot to explore the structure and the content of a site
2. Using Web server log files to retrieve client activity and resource usage

7.3 Content Analysis

The Site Analyzer Content Analysis function enables you to perform the following site management functions:

- Check that the site content contains the correct information

This includes verifying that no part of the page is missing and the page is the correct version.

- Imposing site policies

This includes verifying that each page follows site standards such as metatags and checking links to external sites.

- Forcing site security
Each secure page is properly secured.
- Creating site visualization to show content links and site structure
- Checking content size and its transfer rate
- Checking for broken links
- Ensuring that the content is accessible by checking its HTTP status
- Partitioning the site to provide sub-site views that may be useful for detailed analysis

The hyperlink structure characterizes Web site documents. There are several ways to specify a link in a Web document. The following list summarizes several link types in a static HTML environment:

- <LINK>, <A> and <AREA> tags with an HREF attribute explicitly specify the link to other documents.
- , <APPLET>, <EMBED> and <FRAME> tags specify the location of the Web resources.
- <SCRIPT> and <FORM> specify the location of procedures such as using CGIs or servlets.
- FTP and MAILTO protocols specify a link to other resources using specific protocols.

Currently the Site Analyzer can't analyze dynamic links created using dynamic contents techniques such as with servlets, ASP, and JSP.

In a hyperlink structure, a referenced document may in turn reference links to other documents. The Content Analysis function can trace the links several layers deep. The links may also go to external sites.

For a complex Web site, the amount of content and links can grow very fast. Therefore, it is advisable to partition the site's structure into several sub-sites and analyze each one by one.

Content Analysis uses a Web robot to explore the site's contents and structures. A Web robot is a program that automatically travels down the hypertext structure. Upon reading a Web document, the robot can perform specified tasks and check rules, such as calculating file sizes. Then, it goes to every link in the document. This process continues until a specified time limit or a maximum amount of resources is exceeded.

There is a Robot Exclusion Protocol to indicate which part of the site should not be visited by the robot. The Robot Exclusion Protocol is a standard that is well documented on the robot mailing list (<http://info.webcrawler.com/mailling-lists/robots/info.html>). It is implemented using a file, robot.txt, which is located under the site URL. The file contains two types of statements:

1. User-agent: * | <robot_name> to indicate for which robots the rule should apply. An asterisk (*) indicates all robots.

2. Disallow: <disallowed_URL> to indicate which sites should not be visited by the robot specified in the user-agent.

For example, the following lines in `http://wsnt00.itso.ral.ibm.com/robot.txt` ban every robot from visiting `http://wsnt00.itso.ral.ibm.com/web/classes`:

```
# A comment beginning with *
User-agent: * # This rule applies to any robot
Disallow: /web/classes # Don't detect my classes
```

You can specify a set of analysis tasks and rules for controlling Web robot operations, which include:

- Starting URLs for analysis (Analysis Root URL).
- Analysis depth limit, either in maximum link depth or maximum number of resources to analyze. If either one of the conditions is satisfied, the analysis stops.
- Obey Robot Exclusion Protocol as specified in the site's `robot.txt`.
- Specific actions, such as calculating resource size or checking metatags in Web pages.
- Exclude or include URLs that are to be included in the analysis.

Content Analysis records the time required for accessing a specific resource. It is possible that a Web resource is not accessible. In this case, you can specify a time limit to continue accessing other resources. The analysis will keep trying to access the inaccessible resource until a global time limit is reached. The global time limit is the time when the whole analysis process will stop.

The analysis also records the transfer rate for loading a specific resource. Note that the transfer rate is not a statistical average, but only the value recorded during the analysis.

The Site Analyzer analyzes the following types of contents:

- Web pages
- Applets
- CGI Scripts
- Images
- Mail
- ftp

7.3.1 Using Content Analysis

To create, edit, delete, and run Content Analysis tasks you should be a Site Analyzer Administrator. A client can view only the result using Site Surveyor or Reports.

7.3.1.1 Creating a New Content Analysis

To create a Content Analysis task, perform the following steps:

- In the Site Analyzer Project Center window, select the project with which you want to work.

- Select **File -> New -> Analysis** to bring up the New Analysis wizard as shown in Figure 370 on page 397.
- Select **Create a Content Analysis** and click **Next**.
- The next dialog box asks for the analysis task's name and the analysis root URL (Figure 371 on page 397). Supply a unique analysis name and the starting analysis URL in your site.
- You can either click **Finish** or choose to configure advanced parameters by clicking the **Advanced** button.

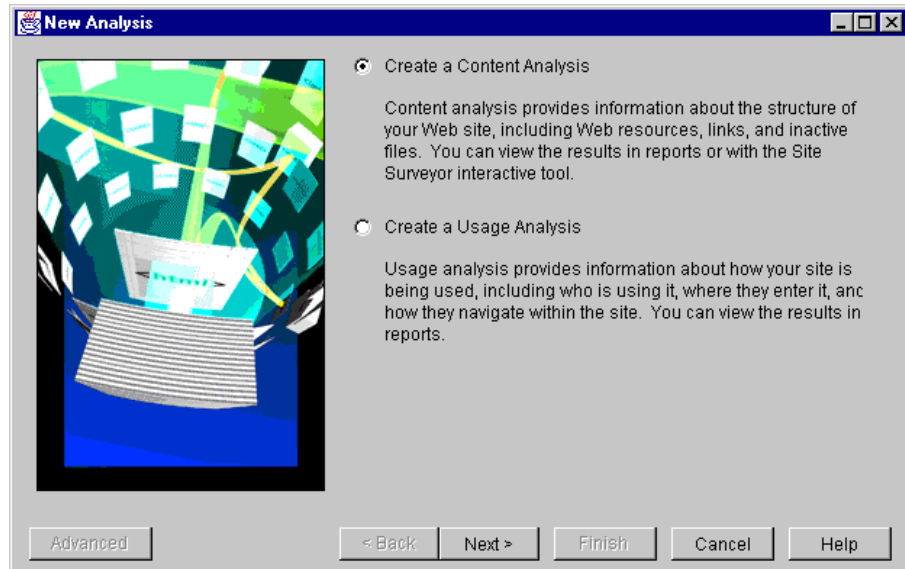


Figure 370. Create a New Content Analysis

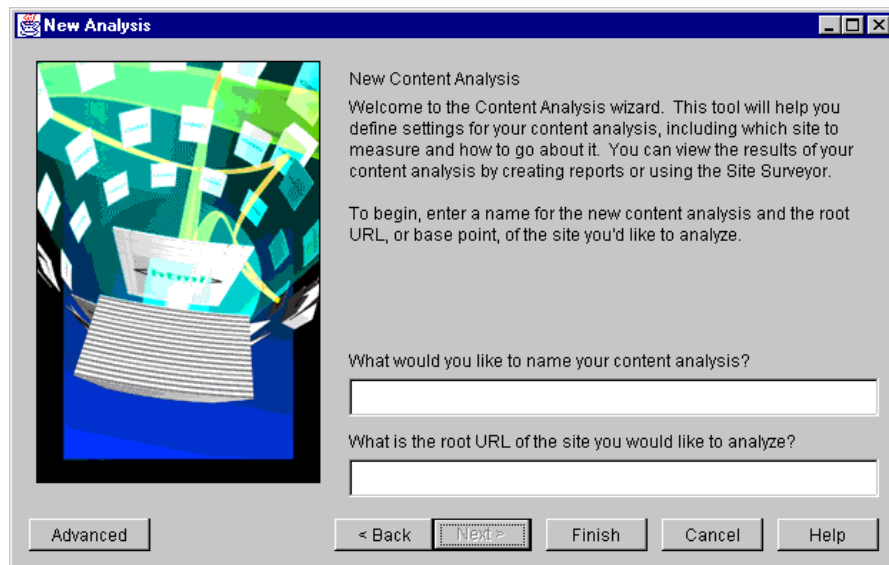


Figure 371. Specifying the Name and the URL for Content Analysis Task

7.3.1.2 Setting Content Analysis Advanced Parameters

To set advanced parameters for a Content Analysis task, you can click either the **Advanced** button during analysis creation or right-click the analysis row, followed

by **Edit** on the analysis contextual menu. This will bring up the Advanced Content Analysis dialog box as shown in Figure 372.

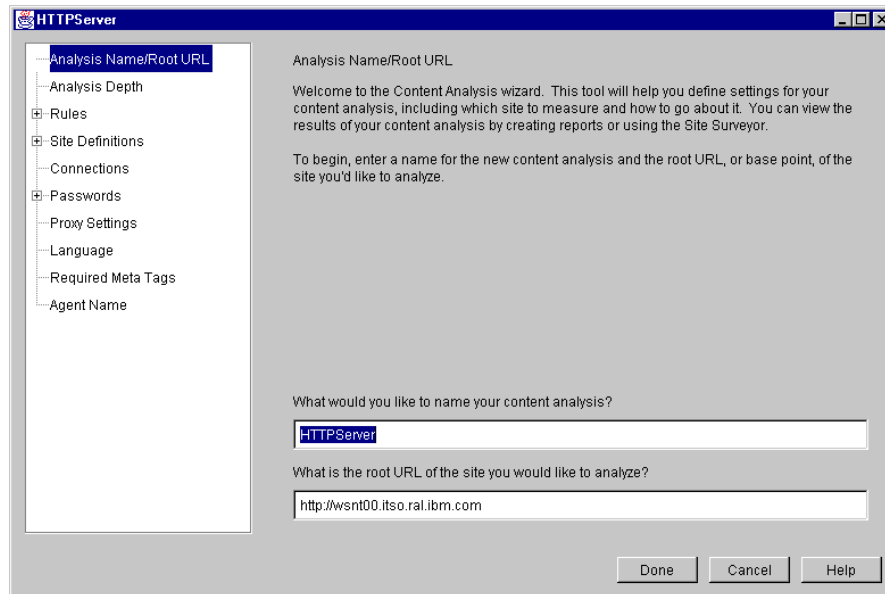


Figure 372. Content Analysis Advanced Settings

In this dialog box you can navigate using a parameter tree in the left frame. Table 35 summarizes the Content Analysis advanced settings:

Table 35. Content Analysis Advanced Setting

Parameter Name	Description
Analysis Name	The name of the analysis.
Root URL	The URL where the analysis will start.
Analysis Depth	Limit how far the analysis will go. It is specified either by maximum link depth or maximum number of resources. If either one of these conditions is satisfied, the analysis will stop.
File Size Calculations	Ask the Web robot to calculate Web resource size. Specify to calculate either aggregate page size, image size, applet size, or embedded object size.
Restrictions	Ask the Web robot to obey the Robot Exclusion Protocol and verify external links. The external links are defined in the site definition parameters.
Site Definitions	Specify which sites are to be analyzed by including the site URL, host name, and domain in Included Lists, or exclude host name and domain in the Excluded Lists.
Connections	Specify time limit to wait for each access before accessing another resource. Specify global time limit up to when the whole analysis task should stop.
Passwords	Specify user ID and password for password-protected pages. Add user IDs and passwords into User passwords list. Specify any key ring file that contains public keys, private keys, and certificates.

Parameter Name	Description
Proxy Settings	If your site uses firewalls, enter proxy and SOCKS settings for this analysis.
Language	Specify language settings to display language information for multi-language sites.
Required Meta Tags	Specify metatags that should exist in HTML pages.
Agent Name	To identify which hit is from the Site Analyzer. A hit (request) in a site is recorded by the Web browser in a log file. This log file usually includes the source (agent) of the hit. The agent name is used by Usage Analysis to exclude Site Analyzer-owned hits.

7.3.1.3 Running Content Analysis

Now you can run the Content Analysis tasks that you have created. To run this analysis task, right-click the analysis row in the right frame, and choose **Run Now** on the selection menu.

As you can see, the status column indicates that the analysis is running. It will run until it reaches a specified depth limit or maximum number of resources or global time limit. If you specify the depth limit as too large, the analysis may run for a very long time. Obviously, the analysis complexity will grow exponentially over the depth limit.

As the analysis finishes, it shows only the completion status of the analysis. You have to view the result using separate Site Analyzer functions.

7.3.1.4 Content Analysis Output

The result of Content Analysis is stored in the Site Analyzer database. There are several ways to view this result:

- View the site map visually by using Site Surveyor (see 7.5, “Site Surveyor” on page 404 for how to use the Site Surveyor).
- Query specific content information by using Quick Find.
- Create a user-defined report. See 7.6, “Reports and Details” on page 406 for how to create report elements and reports.

7.4 Usage Analysis

Usage Analysis can be used for analyzing:

- The client environment, in terms of the client’s browser and platforms
- Client activity

The analysis records client activity in a session (a visit). A visit starts with the first hit on the site up to the point that the client is idle for a pre-determined amount of time. During a client visit, the analysis can gather a lot of valuable information, such as hits per visit, entry or exit point, and duration per visit.

- Response

This can indicate the success rate of requests to a certain Web resource. The analysis also can differentiate errors due to an invalid request by the client, such as clicking the **Stop** button on the browser.

- Resource usage

The resource can be specified by a name or by category.

- Referral

Is the site from where the client makes the request. It is useful for counting advertisement hits from a referring site.

- Entire site usage

The analysis measures this information in units: hits, page view, visit, duration, and bytes transferred. The page view counts hits for the page itself. It does not count hits for items on the page. The analysis also can calculate totals (sums) or averages of a certain measurement.

7.4.1 Web Server Log Files

With each client request, the Web server logs the requests and its response status. The Usage Analysis uses these log files for analyzing Web usage. The Usage Analysis will try to mine any information that can be derived from the log file.

Most Web servers can store log information into the Common Log Format as originally defined by the NCSA. This log format distributes log information into three log files: access, agent and referer files.

The access log file has the following syntax:

```
clienthost authuser [date] "request" status length
```

where:

- `clienthost` is the client IP address or host name
- `authuser` is the user ID as entered by the client
- `[date]` is the date, time and GMT; by default it is the client local time
- `"request"` specifies the request method and the Universal Resource Identifier (URI) for corresponding Web resources
- `status` is the HTTP status code in the Web server response to the client
- Response length is in bytes

As an example, the access log file contained:

```
9.24.105.244 - - [03/May/1999:09:36:35 -0500] "GET /fountainpen3.jpg HTTP/1.0"
200 6357
9.24.104.68 - - [13/May/1999:09:37:36 -0500] "GET
/homepage?OpenElement&FieldElemFormat=gif HTTP/1.0" 404 282
```

The agent and referer files capture HTTP header "user-agent" and "referrer" respectively. The agent specifies client browser type. The referer specifies from where the request was made. As an example, the agent log file contains:

```
[03/May/1999:09:36:35 -0500] "Mozilla/4.0 (compatible; MSIE 4."
[03/May/1999:09:37:36 -0500] "Mozilla/4.51 [en] (WinNT; I)"
...
```

As an example, the referer log file contains:

```
[03/May/1999:09:36:35 -0500] "http://wsnt00/"
```

```
[03/May/1999:09:37:36 -0500] "http://wsnt00.itso.ral.ibm.com/"  
...
```

Some Web servers can use other formats, such as the W3C extended log file format (see <http://www.w3.org/TR/WD-logfile>) or a custom format.

In any case, the Usage Analysis will use Common Log Format fields, plus user-agent and referrer as the minimum required fields. You should configure your Web server log files to include these fields.

7.4.2 Using Usage Analysis

To create, edit, delete, and run Usage Analysis tasks, you should be the Site Analyzer Administrator. A client can view only the results by using reports.

7.4.2.1 Creating a New Usage Analysis

To create a Usage Analysis task, perform the following steps:

- In the Site Analyzer Project Center window, select the project with which you want to work.
- Select **File -> New -> Analysis** to bring up the New Analysis wizard as shown in Figure 373 on page 401.
- Select **Create a Usage Analysis** and click **Next**.
- The next dialog box asks for the Usage Analysis task name. Supply a unique analysis name, then click **Next**.

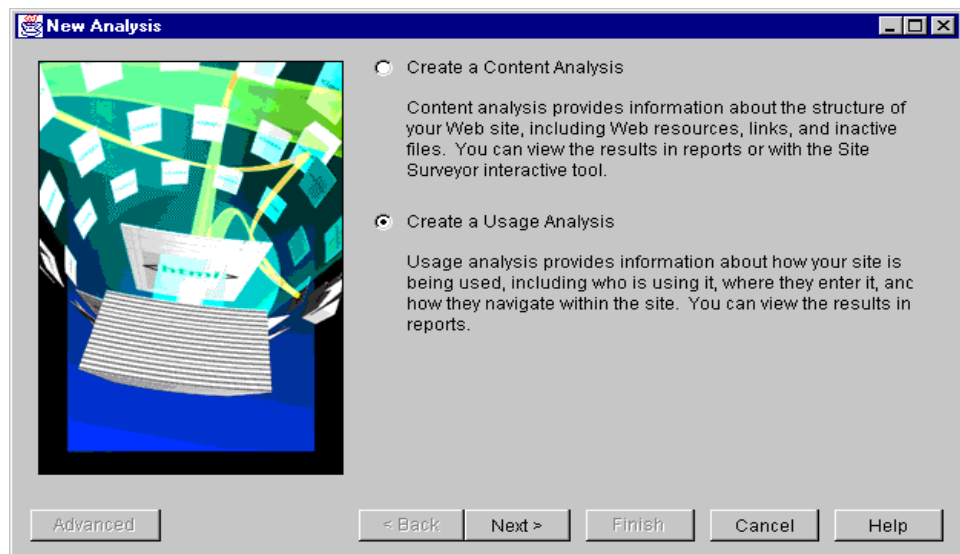


Figure 373. Create a New Usage Analysis

- Add Web server log files that you want analyzed (see Figure 374 on page 402). Click the **Add** button (the left-most button below the log file list). This will bring up the Specify Log Files dialog box (see Figure 375 on page 402). On the first tab select an appropriate log file format. On the second tab, specify the log file name. Then click **OK**.

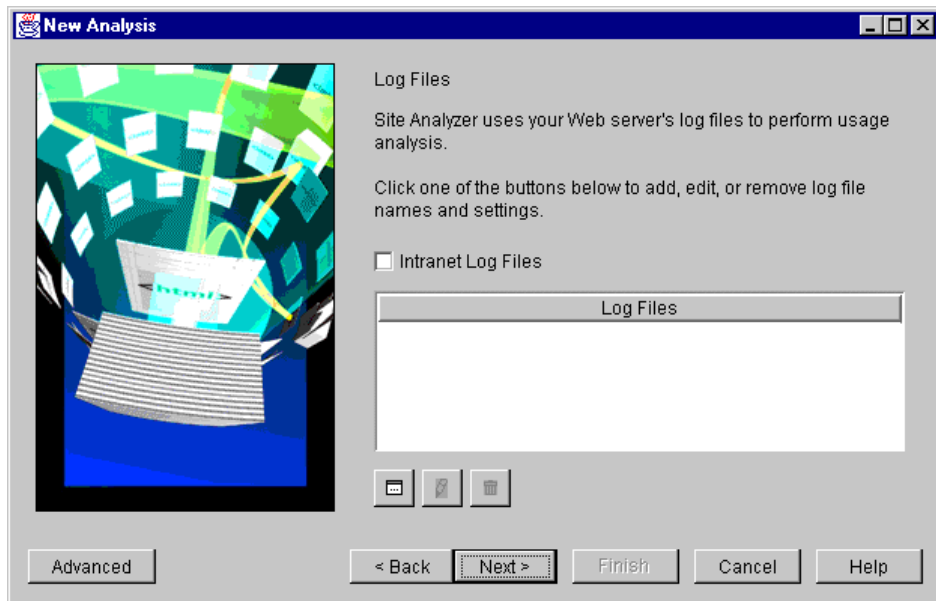


Figure 374. Specify Log Files Dialog Box (1/2)

- In the next dialog box, specify your site hostnames (see Figure 376 on page 403). This is useful to filter out requests from your own hosts.
- Click **Finish** to create a new Usage Analysis.

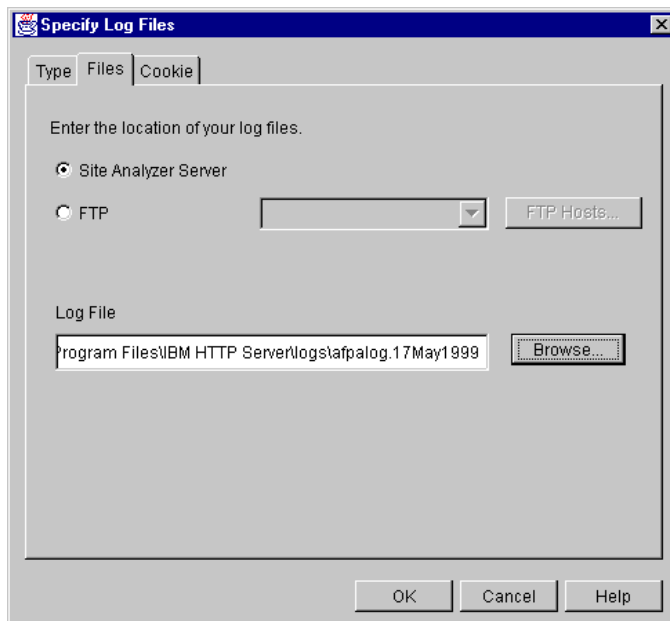


Figure 375. Specify Log Files Dialog Box (2/2)

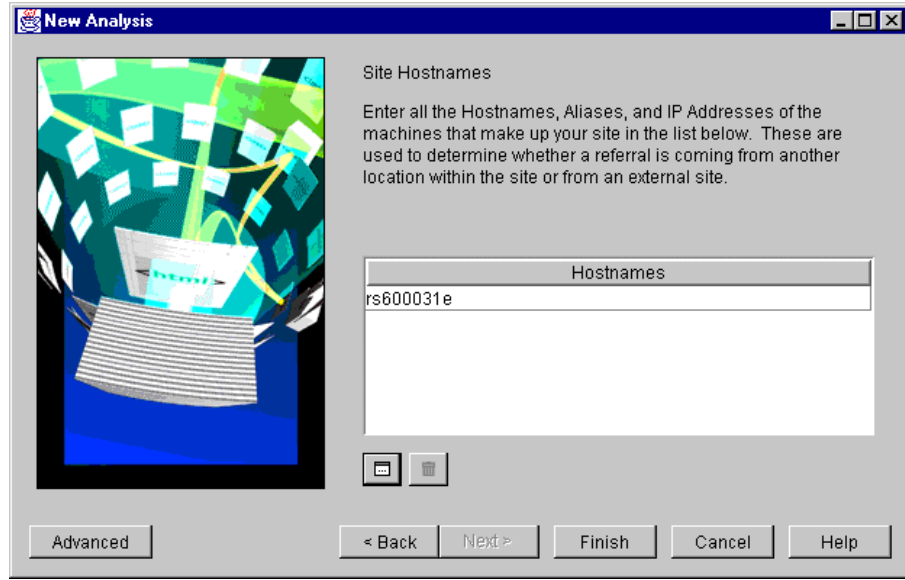


Figure 376. Site Host Names Dialog Box

7.4.2.2 Setting Usage Analysis Advanced Parameters

To set advanced parameters for a Usage Analysis task, you can click either the **Advanced** button during analysis creation or right-click the analysis row followed by selecting **Edit** on the analysis contextual menu. This will bring up the Advanced Usage Analysis dialog box as shown in the following window:

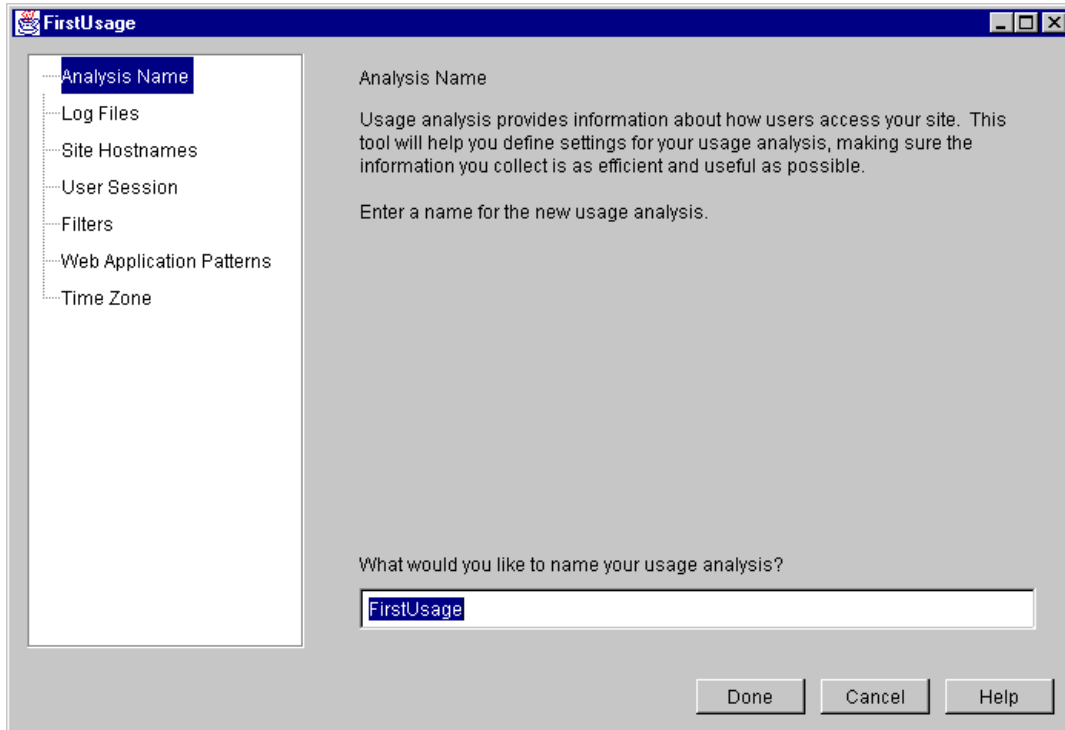


Figure 377. Usage Analysis Advanced Settings

In this dialog box, you can navigate using a parameter tree in the left frame. Table 36 summarizes the Usage Analysis advanced settings:

Table 36. Content Analysis Advanced Setting

Parameter Name	Description
Analysis Name	The name of the analysis.
Log Files	Web server log files to be analyzed.
Site Hostnames	Your site host names. This is useful to identify which requests are from internal sites and which requests are from external sites.
Session Length	Time limit for a user session time.
Filters	Exclude artwork requests or requests from specified IP addresses.
Web Application Patterns	Count only hits for a specified application and its parameter pattern.
Time Zone	Convert time zone information in the requests into a specified time zone.

7.4.2.3 Running Usage Analysis

Now you can run the Usage Analysis tasks that you have created. To run the analysis task, right-click the analysis row in the right frame, and choose **Run Now** on the selection menu.

As you should be able to see, the status column indicates that the analysis is running. It may take some time to read the Web server log files and write the results into the Site Analyzer database.

7.4.2.4 Usage Analysis Output

The result of the Usage Analysis is stored in the Site Analyzer database. To view the result:

- Create a report element that contains only one measurement topic. It is useful for a quick view for a specific measurement.
- Create several report elements and combine them into a report.

7.5 Site Surveyor

The Site Surveyor shows the result of a Content Analysis visually. It displays:

- Site structure in a tree view starting from the analysis root URL. You can expand or collapse sub-trees to provide necessary details. Each tree leaf corresponds to a resource (page, image, applet, etc.) or a link.
- Information about resource and link.

The Site Surveyor provides the following resource information:

- Resource type: HTML, applet, embedded object, CGI script, etc.
- Resource URL
- Resource Title, such as HTML <TITLE> page title
- HTTP Status returned when the analysis accesses the resource

- Transfer rate when the analysis accesses the resource
- The size of the resource
- The aggregate size, which is the sum of the page size and all objects in it
- The last modified date that can indicate the version of the resource
- Expiration date as indicated in the "META" metatag.

The Site Surveyor provides the following link information:

- Link text
- Link type according to the way the link is put in the HTML; for example, <LINK> specifies hyperlink, specifies image, and so on
- Broken link, which can be a resource that does not exist when the analysis tries to visit it, or a time-out hyperlink such as those caused by a busy server

To use Site Surveyor, run a Content Analysis. After the analysis has stopped, right-click the **Content Analysis** and select **View with Site Surveyor**.

The Site Surveyor will read Content Analysis results from the database, and show the content according to resource type and their links.

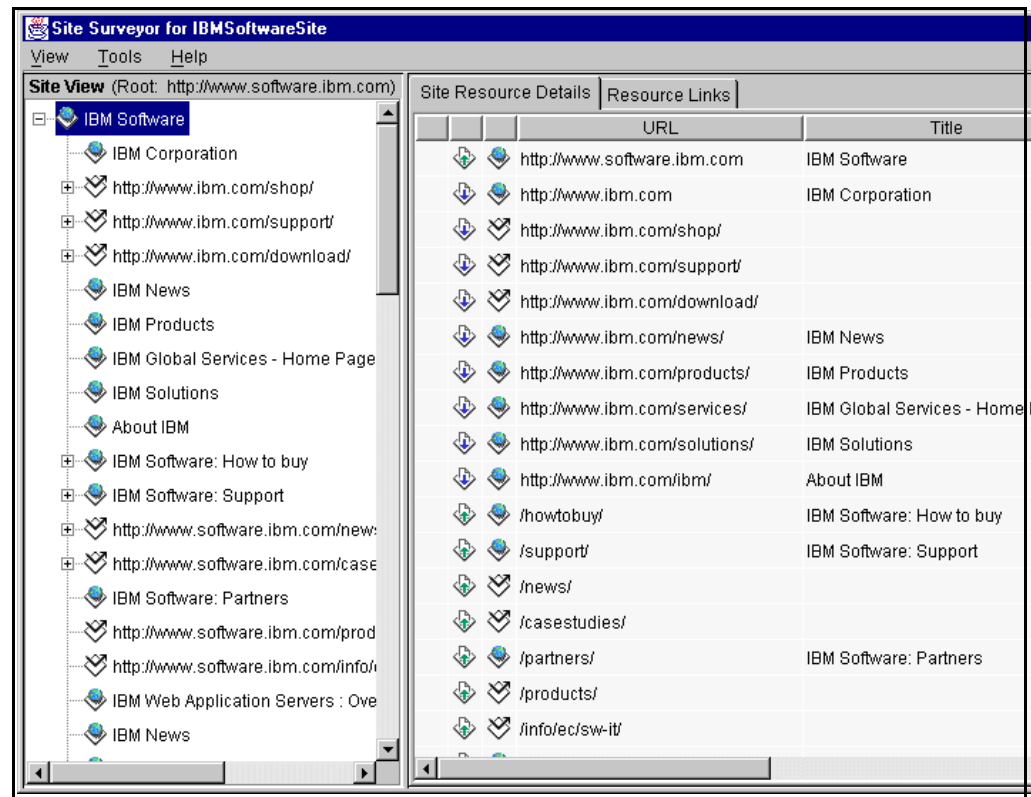


Figure 378. Site Surveyor Main Window

For example, Figure 378 on page 405 is the result of Content Analysis on the IBM Software site with <http://www.software.ibm.com> as the analysis root URL. In the Site Surveyor left frame, there is a site structure tree with IBM Software as the root. In this case, the IBM Software is an HTML page. On the right frame there are two tab sheets. The Site Resource Details tab shows all resources that can

be linked from the HTML page. The Resource Links tab (Figure 379 on page 406) shows link information for each link. Notice that in this figure, in the left-most column of Resource Link Tabs, the link to `/cgi-bin/listbox-redirect.pl` is broken.

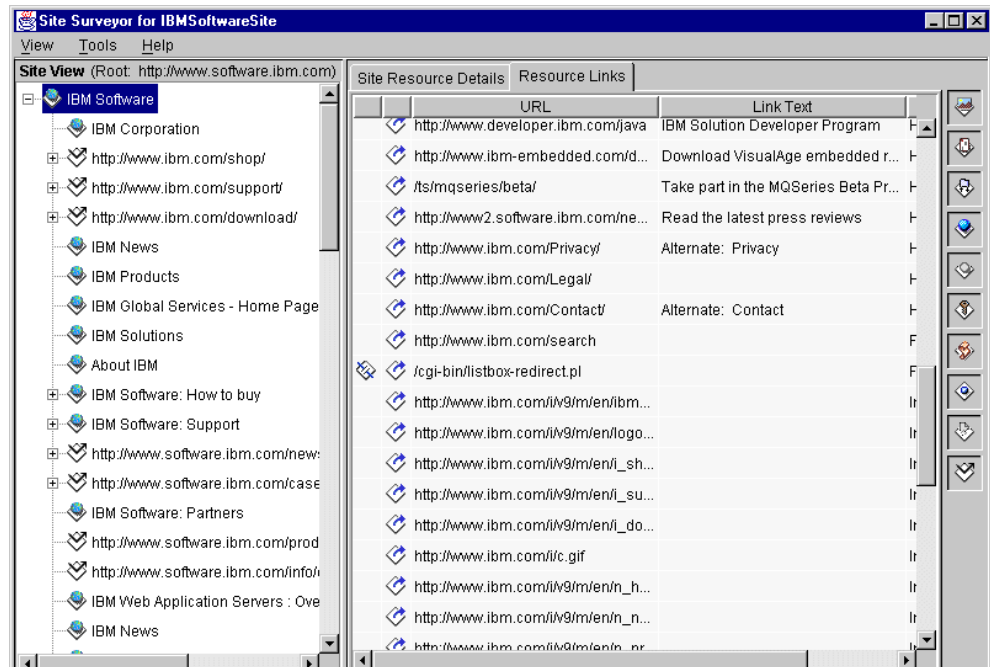


Figure 379. Site Surveyor Indicates a Broken Link on IBM Software Site

7.6 Reports and Details

The Site Analyzer can present analysis results as reports. The report is built upon several report elements. Each report element represents an analyzed item to be included in the report.

Either the Site Analyzer Administrator or client user can create report elements and reports. Site analyzer provides report element and report wizards that are easy-to-use and self-explanatory.

7.6.1 Report Element

The report element setting consists of:

- Type, which is the measured item during analysis
- Measurement unit
- Filter, for creating a more specific result
- Presentation format, which can be in tabular format or in a graphical chart

In Content Analysis, analyzed item types are:

- Pages
- Links

In Usage Analysis, analyzed item types are:

7.6.1.1 Content Analysis Report Element

A Content Analysis stores its results in the Site Analyzer database. To view the result, you should use either Site Surveyor, or Quick Find, or create a report element. You can create several report elements and include them in a single report. You can specify the report element even before the analysis is run.

To create a report element on a Content Analysis complete the following steps:

- From the Menu Bar, Select **File -> New -> Content Report Element** to invoke the Report Element Wizard for Content Analysis as shown in Figure 380:

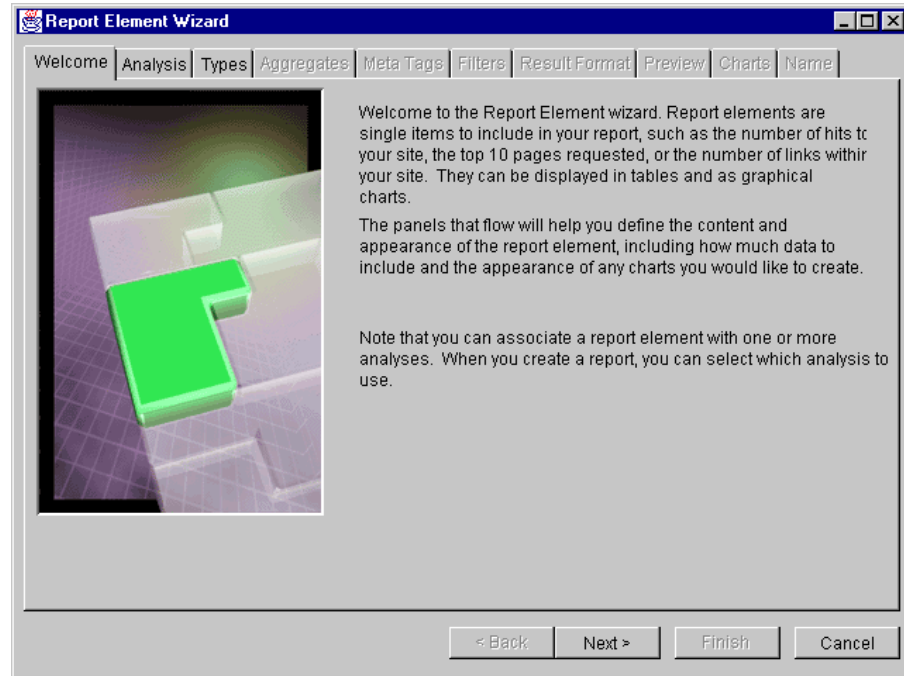


Figure 380. Report Element Wizard for Content Analysis

- The Wizard consists of several tab sheets. Table 37 summarizes Content Analysis report element settings:

Table 37. Content Analysis Report Element Settings

Setting	Description
Analysis	Select a Content Analysis to use for this report element.
Type	Items that will be reported, which can be pages or links.
Aggregates	How the item is measured or grouped.
Filter	Specify query condition in more detail by using item parameters.
Result Format	Specify maximum rows returned in the report. Specify sort key and sort order.
Preview	To view the sample of the report.
Chart	To create a chart based on the item data.

Setting	Description
Name	The name of this report element.

- After completing the Report Element Wizard page, you can click to create a new Content Analysis report element.

You can always change report element settings by right-clicking the analysis in the right frame and selecting **Edit**.

7.6.1.2 Usage Analysis Report Element

To view the result of a Usage Analysis, you should create several report elements and combine them in a report. You can specify the report element even before the analysis is run.

To create a report element on a Usage Analysis complete the following steps:

- From the Menu Bar, Select **File -> New -> Usage Report Element** to invoke the Report Element Wizard for Usage Analysis.
- After completing the Report Element Wizard page, you can click to create a new Usage Analysis report element.

You can always change report element settings by right-clicking the analysis in the right frame and selecting **Edit**.

7.6.2 Report

After creating several report elements, you can create a report that combines the report elements.

To create a report complete the following steps:

- From menu bar, select **File -> New -> Report** to invoke the Report Element Wizard.
- The Wizard consists of several tab sheets. In the Report Elements tab select report elements to be included in this report.
- In the Order tab sheet, you can set the order of report element appearance in the report.
- You can add a glossary of terms specified in the report. To do this, click any term on the left frame of the tab sheet.
- On the last tab, enter the name for this report. Then, click **Finish** to create the report.

7.6.2.1 Generating the Report

The report generation extracts analysis data from the database, performs calculations according to the report specification and creates an HTML document as a report. This process also allows you to do the final cosmetic artwork for the report, which includes adding a logo and selecting a table style.

In addition, you can specify the time range for the report and whether the report should be dynamically generated.

The dynamically generated option performs analysis data extraction when the report is requested by a client. A report of this can be published to a Web server.

To generate a report, select the report item from the Project tree or from the report list on the right frame. Right-click the item and select **Generate Report**. This will bring up the Report Generator Wizard as shown in Figure 381:

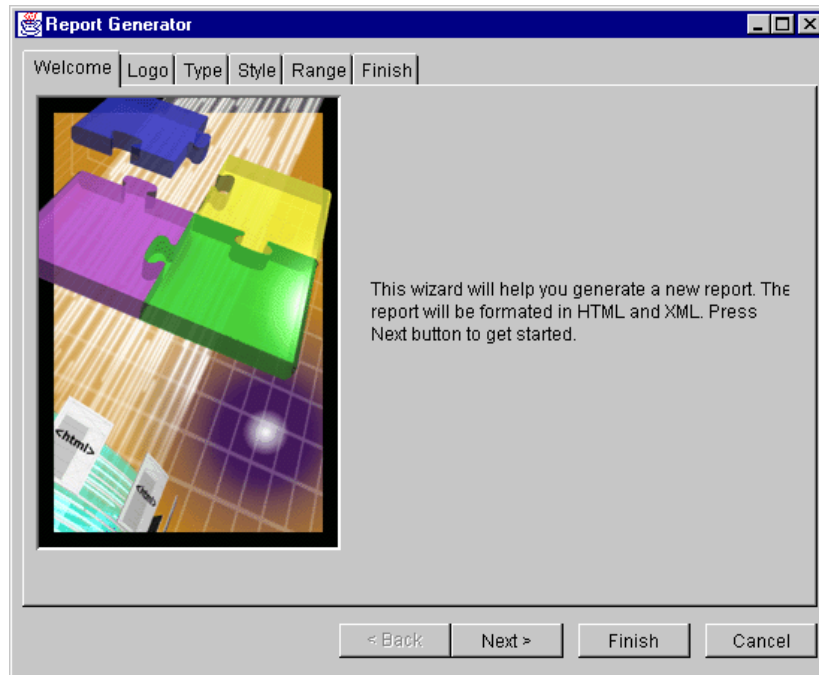


Figure 381. Report Generator Wizard

As the report is in HTML format, you can view it using a browser. You can do it by selecting the report item on the Project tree or on the report list in the right frame and continue with the following:

- Right-click the item and select **Show**. This will automatically call a Web browser to display the report. The report HTML documents will be kept locally in the <SiteAnalyzerRoot> directory.
- Right-click the item and select **Publish**. The Site Analyzer will ask for a publish destination (Figure 382). You can put the result in another Web server.

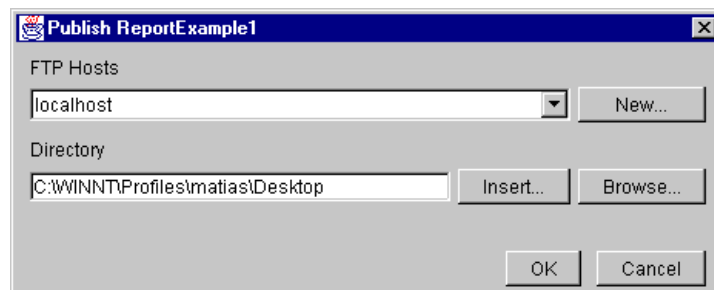


Figure 382. Specifying Publish Destination

Chapter 8. Problem Determination

In this chapter we look at the functions that are available in WebSphere and other products that help with the problem determination process.

8.1 WebSphere Log Files

This section shows you what logs are available in WebSphere, what you would expect to see in them, what they are good for and how to enable them. There is some overlap with other sections in this chapter as other functions can also produce logs, so some sections simply contain references to sections later in the chapter.

8.1.1 Overall Log Structure

Table 38 shows some general information on the logs available in WebSphere. The Relative Directory column shows the directory relative to the root log directory at <Server Root>/logs. Note that these file names are configurable if desired; the table values represent the default values.

Table 38. Log File General Information

Log File Name	Relative Directory	Description	Described in
jvm_stderr.log	/	Standard error output from the JVM.	8.1.2, "The JVM Standard Error Log" on page 412.
jvm_stdout.log	/	Standard output from JVM.	8.1.3, "The JVM Standard Output Log" on page 413.
oop_native.log.ERROR	/	Error messages from the native code portion of the out-of-process engine. Seldom used.	N/A
oop_native.log.INFORM	/	Informational messages from the native code portion of the out-of-process engine. Seldom used.	N/A
apache.log.INFORM.*	/	IBM HTTP Server informational messages.	8.1.5, "The IBM HTTP Server Error Log" on page 414.
apache.log.ERROR.*	/	IBM HTTP Server error messages.	8.1.5, "The IBM HTTP Server Error Log" on page 414.
apache.log.TRACE.*	/	Trace log for IBM HTTP Server.	N/A

Log File Name	Relative Directory	Description	Described in
websphere_trace.log	/	Trace information generated by WebSphere trace monitors.	8.1.6, "The WebSphere Trace Log" on page 414.
error_log	/servlet	General error messages for the servlet engine. Hardly ever used.	N/A
event_log	/servlet	General event messages for the servlet engine. Hardly ever used.	N/A
error_log	/servlet/adminservice	Error messages from the WebSphere administration application.	8.1.7, "The Servlet Admin Service Error Log" on page 415.
event_log	/servlet/adminservice	Informational messages from the WebSphere administration application.	8.1.8, "The Servlet Admin Service Event Log" on page 416.
access_log	/servlet/adminservice	Log of access requests to the WebSphere administration application.	8.1.9, "The Servlet Admin Service Access Log" on page 416.
error_log	/servlet/servletservice	Error messages from the servlet service.	8.1.10, "The Servlet Service Error Log" on page 417.
event_log	/servlet/servletservice	Informational messages from the servlet service.	8.1.11, "The Servlet Service Event Log" on page 418.
access_log	/servlet/servletservice	Log of access requests to the servlet service.	8.1.12, "The Servlet Service Access Log" on page 418.
trace.*.log	/trace	Log of activity in the WebSphere server engine.	8.1.13, "The WebSphere Engine Tracing Log" on page 418.

8.1.2 The JVM Standard Error Log

The JVM standard error log is probably the most useful log of all the WebSphere logs. In this log the standard error output from the main WebSphere servlet Java process appears. Standard output and standard error output from the EJS Java processes are available using the EJS, PNS and LSD tracers (see 8.3.1, "Tracers" on page 425). The format of the entries in this log depends on the application writing them. However, stack traces from *caught* and *uncaught* Java exceptions are common. To direct application output to this log, use the methods

of the static `java.lang.System.err` object in your Java code. This log is always enabled.

8.1.3 The JVM Standard Output Log

The JVM standard output log is the second most useful one in WebSphere. The standard output log contains all of the output written by Java applications to standard output using the methods of the static `java.lang.System.out` object. The format of this log is application dependent. Use this log to inspect debugging and informational messages from your Java applications writing to `System.out`. This log is always enabled.

8.1.4 The IBM HTTP Server Information Log

The IBM HTTP Server information log contains informational and error messages generated by the IBM HTTP Server included with WebSphere. Each time the server is re-initialized a new log is generated with a different number used as the last component of the file name. The major part of each of these logs is taken up with messages showing the properties with which WebSphere is initialized. This is useful in confirming the parameter values that a particular invocation of WebSphere is actually working with.

Figure 383 on page 414 shows some sample output from the information log with lines containing information on the property values. Each line contains the following components:

- An ID number specific to the particular invocation of WebSphere. This is the same number used as a suffix to the log file name.
- A timestamp.
- The word Property.
- The property name and value separated by an = sign.

For information on the property names and what the possible values are, see the WebSphere documentation at:

<http://<Your Server Name>:9527/doc/howto/itedit.html>

```

id 95 Mon May 24 08:04:52 1999 - Property java.classpath =
d:\JDK11~1.7\lib\classes.zip;d:\WEBSPH~1\APPSE~1\classes;d:\WEBSPH~1\APPS
ER~1\web\classes;d:\WEBSPH~1\APPSE~1\lib\jsdk.jar;d:\XMLTOO~1\XML4J_~1\xm
l4j.jar;d:\XMLTOO~1\LOTUSX~2\lotusxsl.jar;D:\SQLLIB\java\db2java.zip

id 95 Mon May 24 08:04:52 1999 - Property ose.server.library.msjvm =
asoutsm.dll

id 95 Mon May 24 08:04:52 1999 - Property ose.outofproc.transport.type =
local

id 95 Mon May 24 08:04:52 1999 - Property java.asyncgc = true

id 95 Mon May 24 08:04:52 1999 - Property ose.library.out.jni = asoutc.dll

id 95 Mon May 24 08:04:52 1999 - Property ose.outofproc.hostname =
localhost

id 95 Mon May 24 08:04:52 1999 - Property java.path = d:\JDK11~1.7\bin

id 95 Mon May 24 08:04:52 1999 - Property server.name = servlet

id 95 Mon May 24 08:04:52 1999 - Property ose.trace.file = trace.PID.log

```

Figure 383. Sample Output from `apache.LOG.INFORM`

8.1.5 The IBM HTTP Server Error Log

The IBM HTTP Server error log contains error messages generated by the IBM HTTP server included with WebSphere. The format and content is similar to that of the information log (see 8.1.4, “The IBM HTTP Server Information Log” on page 413), but the messages are restricted to those describing error conditions. These messages are also included in the information log.

The error log is useful for diagnosing problems where the Web server or the WebSphere services fail to start for some reason. On Windows NT we frequently encountered the situation where the NT service database was locked as the system booted leading to the servlet service not starting and placing the following message in the log:

```

id 95 Mon May 24 08:04:52 1999 - Error : nt service data base is locked,
can not start ose.

```

The solution was to stop and restart the Web server, which then started the servlet service correctly.

8.1.6 The WebSphere Trace Log

The WebSphere trace log contains the output of the different tracing functions described in 8.3, “Tracing” on page 424. Each log entry is the result of an event picked up by one of the tracing monitors and can contain any of the following six components:

1. A timestamp
2. The name of the tracer that is the source of the event
3. The ID of the thread that caused the event
4. The severity level of the tracer message
5. Any exception generated by the event
6. The message text

The actual components of each message generated in the log are configured using the trace output handler settings (see 8.3.4, “Setting Trace Properties Using the Debug.properties File” on page 428). By default, each message contains the timestamp, tracer name, severity and message in that order. The trace log is useful for gathering detailed debugging information about different WebSphere components in a selective manner. The settings that determine which events generate messages in the log are configurable. See Table 42 on page 428 and the following explanatory note number 3 on page 430 for details on how to configure this using the levelThreshold setting.

8.1.7 The Servlet Admin Service Error Log

The servlet admin service error log contains messages that relate to errors that occur during the execution of the WebSphere administration service. This includes errors that may occur while running any tracing programs (see 8.3, “Tracing” on page 424). Errors encountered by individual servlets are placed in the servlet service error log (see 8.1.10, “The Servlet Service Error Log” on page 417). The format of each message is a timestamp followed by the error message.

Exceptions are handled slightly differently in that the entire exception stack trace is included surrounded by message lines that delineate the start and end of the exception block. Figure 384 on page 416 shows an exception generated during tracing along with the enclosing exception block start and end lines.

The servlet admin service error log can be used to help diagnose errors encountered while running the WebSphere administration application.

```

[April 20, 1999 10:05:28 PM EDT] START EXCEPTION BLOCK
[April 20, 1999 10:05:28 PM EDT] java.net.SocketException: Socket write
failed: 10053
java.net.SocketException: Socket write failed: 10053
at java.net.SocketOutputStream.write(SocketOutputStream.java:91)
at sun.servlet.http.HttpOutputStream.writeOut(HttpOutputStream.java:485)
at
at sun.servlet.http.HttpOutputStream.flush(HttpOutputStream.java:345)
at java.io.OutputStreamWriter.flush(OutputStreamWriter.java:230)
at com.sun.server.http.FileServlet.copy(FileServlet.java:447)
at com.sun.server.http.FileServlet.writeResponse(FileServlet.java:338)
at com.sun.server.http.FileServlet.sendResponse(FileServlet.java:267)
at com.sun.server.http.FileServlet.service(FileServlet.java:192)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:588)
at
at com.sun.server.ServletManager.callServletService(ServletManager.java:1317)
at
at com.sun.server.ProcessingState.invokeTargetServlet(ProcessingState.java:43
4)
at
at com.sun.server.http.HttpProcessingState.execute(HttpProcessingState.java:9
3)
at com.sun.server.http.stages.Runner.process(Runner.java:77)
at com.sun.server.ProcessingSupport.process(Compiled Code)
at com.sun.server.Service.process(Service.java:229)
at
at com.sun.server.http.HttpServiceHandler.handleRequest(HttpServiceHandler.ja
va:350)
at
at com.sun.server.http.HttpServiceHandler.handleRequest(HttpServiceHandler.ja
va:210)
at com.sun.server.HandlerThread.run(HandlerThread.java:154)
[April 20, 1999 10:05:29 PM EDT] END EXCEPTION BLOCK

```

Figure 384. An Exception Recorded in the Servlet Admin Service Error Log

8.1.8 The Servlet Admin Service Event Log

The servlet admin service event log contains informational messages concerned with the running of the WebSphere administration service. The administration service uses a number of servlets to perform WebSphere administration tasks. It does not contain messages related to the running of individual servlets; these are recorded in the servlet service event log (see 8.1.11, “The Servlet Service Event Log” on page 418). The format of this log is the same as the format of the servlet admin service error log described in 8.1.7, “The Servlet Admin Service Error Log” on page 415.

The servlet admin service event log can be used to monitor the execution of the WebSphere administration service during WebSphere administration.

8.1.9 The Servlet Admin Service Access Log

The servlet admin service access log contains a record for each HTTP request made to the WebSphere administration application as well as any codes returned. It includes requests for all files associated with the servlet

administration service. The log follows the same format as a standard Web server access log (see Figure 385 for an example):

```
9.24.104.154 - admin [13/Apr/1999:13:09:09 -0400] "POST /servlet/admin
HTTP/1.0" 200 1157
9.24.104.154 - admin [13/Apr/1999:13:09:09 -0400] "POST /servlet/admin
HTTP/1.0" 200 912
9.24.104.154 - admin [13/Apr/1999:13:09:10 -0400] "POST /servlet/admin
HTTP/1.0" 200 912
9.24.104.154 - - [13/Apr/1999:13:10:16 -0400] "GET
/doc/whatis/icusrprf.html HTTP/1.0" 200 5175
9.24.104.154 - - [13/Apr/1999:13:10:46 -0400] "GET /doc/howto/itxml4j.html
HTTP/1.0" 200 56626
9.24.104.154 - - [13/Apr/1999:13:10:46 -0400] "GET /doc/howto/asv2.css
HTTP/1.0" 200 2765
9.24.104.154 - - [13/Apr/1999:13:11:22 -0400] "GET /doc/whatis/icxml4j.html
HTTP/1.0" 200 28803
9.24.104.154 - - [13/Apr/1999:13:15:26 -0400] "GET /doc/howto/itxml4j.html
HTTP/1.0" 200 56626
9.24.104.154 - - [13/Apr/1999:13:15:26 -0400] "GET /doc/howto/asv2.css
HTTP/1.0" 200 2765
9.24.104.154 - - [13/Apr/1999:13:17:40 -0400] "GET
/doc/howto/itbullmsg.html HTTP/1.0" 200 6718
9.24.104.154 - - [13/Apr/1999:13:17:40 -0400] "GET /doc/howto/asv2.css
HTTP/1.0" 200 2765
9.24.104.154 - - [13/Apr/1999:13:27:12 -0400] "GET /doc/howto/itpubsv.html
HTTP/1.0" 200 8264
9.24.104.154 - - [13/Apr/1999:13:27:12 -0400] "GET /doc/howto/asv2.css
HTTP/1.0" 200 2765
9.24.104.154 - - [13/Apr/1999:13:27:42 -0400] "GET /doc/howto/itpubsv.html
HTTP/1.0" 200 8264
9.24.104.154 - - [13/Apr/1999:13:27:42 -0400] "GET /doc/howto/asv2.css
HTTP/1.0" 200 2765
```

Figure 385. Sample Output from the Administration Service Access Log

The servlet admin service access log can be used to monitor the access requests that are sent to the WebSphere administration application.

8.1.10 The Servlet Service Error Log

The servlet service error log contains error information generated by the WebSphere servlet service in attempting to run servlets. It includes both errors pertaining to particular servlets and information pertaining to the servlet service itself. System.out and System.err output that is produced by servlets is placed in the JVM standard output (see 8.1.3, “The JVM Standard Output Log” on page 413) and standard error (see 8.1.2, “The JVM Standard Error Log” on page 412) logs respectively.

The format of this log is identical to that used by the servlet admin service event and error logs (see 8.1.7, “The Servlet Admin Service Error Log” on page 415).

This log can be used to help diagnose both the cause of servlet failures and failures of the servlet service. This log will record any exceptions thrown by WebSphere in trying to load or configure servlets (see 3.1, “How to Deploy and

Configure a Servlet” on page 91) so it is a good place to start diagnosing these sorts of problems. It will also contain information relevant to determining the causes of servlet service start failures.

8.1.11 The Servlet Service Event Log

The servlet service event log parallels the format of the admin service event log (see 8.1.8, “The Servlet Admin Service Event Log” on page 416) but the information contained within it pertains to servlets executing in WebSphere and to the servlet service itself. It lets you see when servlets are loaded, when they are called, and with what parameters they are called. This is useful in trying to diagnose why a particular servlet is failing under certain conditions but not others, since it shows you how it is being called. You may want to add some debug writes to your servlet that includes a timestamp so you can match the debug output that your servlet is producing in standard output or standard error to the messages in the event log which already have a timestamp.

8.1.12 The Servlet Service Access Log

The servlet service access log will generally be empty. See your Web server’s access log for file access information, or use one of the servlet tracers described in 8.3.1, “Tracers” on page 425.

8.1.13 The WebSphere Engine Tracing Log

The WebSphere engine tracing log shows detailed information about the activity on the server. Each action that the server takes is recorded. The format of this log is a timestamp and a message.

This log is good for diagnosing classpath problems, since it provides detailed information on which directories and JAR files have been found and verified in the classpath.

This log is not enabled by default. To enable it you need to edit the <Server Root>/properties/bootstrap.properties file and set the `ose.trace.enabled` property near the top of the file to true and then stop and restart WebSphere.

8.2 The Application Server Debug Console

The application server debug console provides a mechanism by which the WebSphere Application Server can be monitored either on an ongoing basis or for debugging purposes. It provides a similar set of tools to those provided under Server Execution Analysis in the WebSphere administration interface with the important inclusion of a console monitor (see 8.2.2, “The Server Console Monitor” on page 419). In this section we look at how to enable the application server debug console. Then we show the different functions provided and what they are used for.

8.2.1 Enabling the Console

To enable the application server debug console you must edit the <Server Root>/properties/server/servlet/debug.properties file and change the line:

```
debug.server.console.enabled=false
```

to

```
debug.server.console.enabled=true
```

In the default file that ships with WebSphere 2.02 Advanced, this line occurs at line 32 in the file. After making this change and saving the file, stop the WebSphere Servlet Service and the HTTP server and then restart them to activate the console. The window shown in Figure 386 on page 419 should appear (minus the output) when the WebSphere Servlet Service starts. If there is a problem then you should check the log files for any possible errors (see 8.1, “WebSphere Log Files” on page 411).

8.2.2 The Server Console Monitor

The server console monitor (shown in Figure 386 on page 419) provides a facility for the monitoring of the output from the different WebSphere tracers (see 8.3, “Tracing” on page 424 for a description of the different tracers) as well as any output sent to System.out or System.err by Java programs. The format of the entries shown in the console depends on the source of the data. Entries from System.out and System.err are in the format specified by the generating program. Output from the WebSphere tracers is in the format specified in the debug.properties file (see 8.3.4, “Setting Trace Properties Using the Debug.properties File” on page 428 for details) but by default includes the name of the tracer producing the message, a number indicating the severity, and the message text.

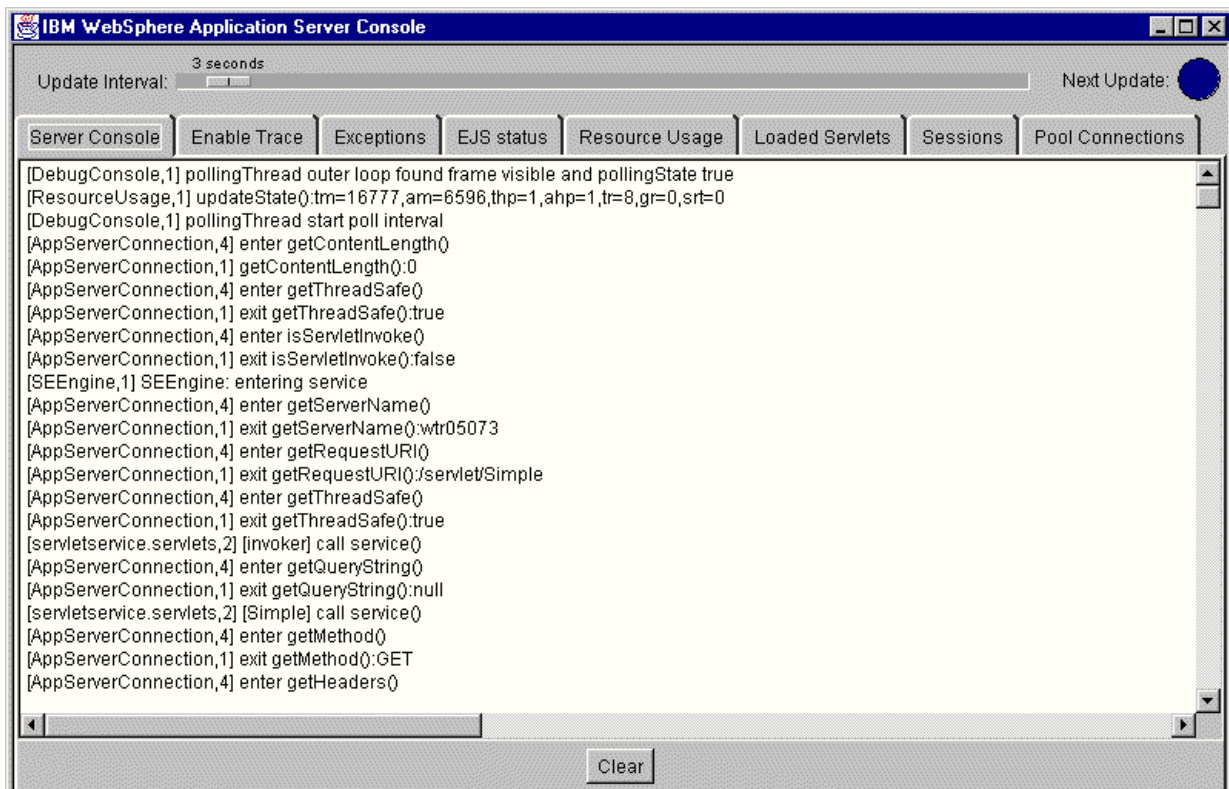


Figure 386. The Debug Console Server Console Monitor Page

During execution the console monitor automatically displays output from the input sources it monitors. If several of the tracers are enabled it can produce significant quantities of output. To clear the current output, click the **Clear** button at the bottom of the page.

8.2.3 The Trace Enabler Page

The trace enabler page provides the ability to modify the settings for the WebSphere tracers. These settings take precedence over the settings present in the debug.properties file (see 8.3.4, “Setting Trace Properties Using the Debug.properties File” on page 428). The trace enabler page is shown in Figure 387:

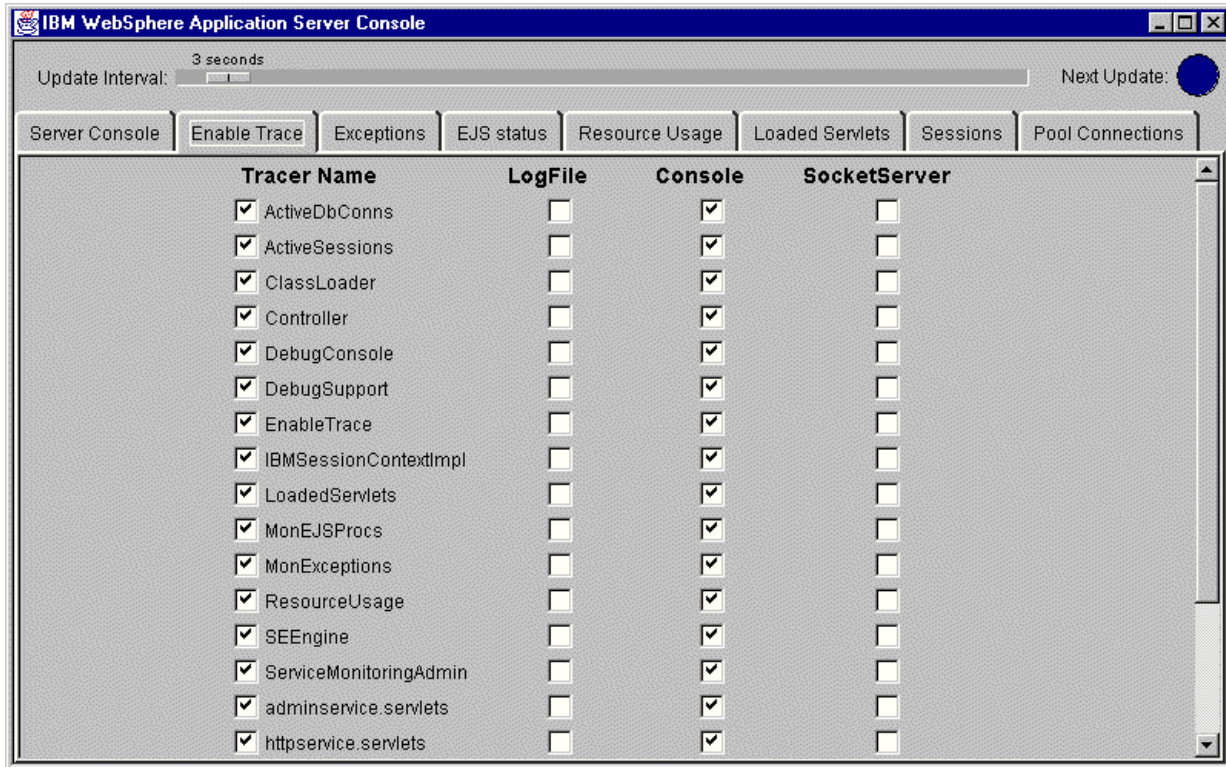


Figure 387. The Debug Console Trace Enabler Page

Each tracer can be turned on by selecting the check box in the Tracer Name column or alternatively all of the tracers can be turned on or off by clicking the **Enable All** or **Disable All** buttons below the column (not shown in the figure). Once a tracer has been turned on, its output must be directed to one of the three trace output destinations for it to be produced. More output destinations will be available if user-defined output destinations have been defined (see 8.3.6, “Creating Your Own Trace Output Handlers” on page 431). This can be accomplished by selecting or deselecting the check boxes in one or more columns to select the desired output destination(s), or by clicking the **Direct All** or **Direct None** buttons below the columns. The LogFile output destination will send output to the file defined in the trace.handler.LogFile.param.filename property of the debug.properties file which is <Server Root>/logs/websphere_trace.log by default. The console output destination will route messages to the server debug console monitor page described in 8.2.2, “The Server Console Monitor” on page 419. The SocketServer output destination directs output to the console provided by the socket server application (see 8.3.3, “Running the Socket Server Trace Console” on page 427). It is important to note that unless both check boxes are enabled, one to enable the tracer and one to direct output to the desired destination, no output will appear.

8.2.4 The Exceptions Monitor

The exceptions monitor provides a facility for inspecting the stack traces of Java exceptions thrown by WebSphere or by user applications running under WebSphere. This method of monitoring exceptions may be preferable to some but we found that monitoring exception data in the log files also worked well and almost never referred to this page. The exceptions monitor is shown in Figure 388:

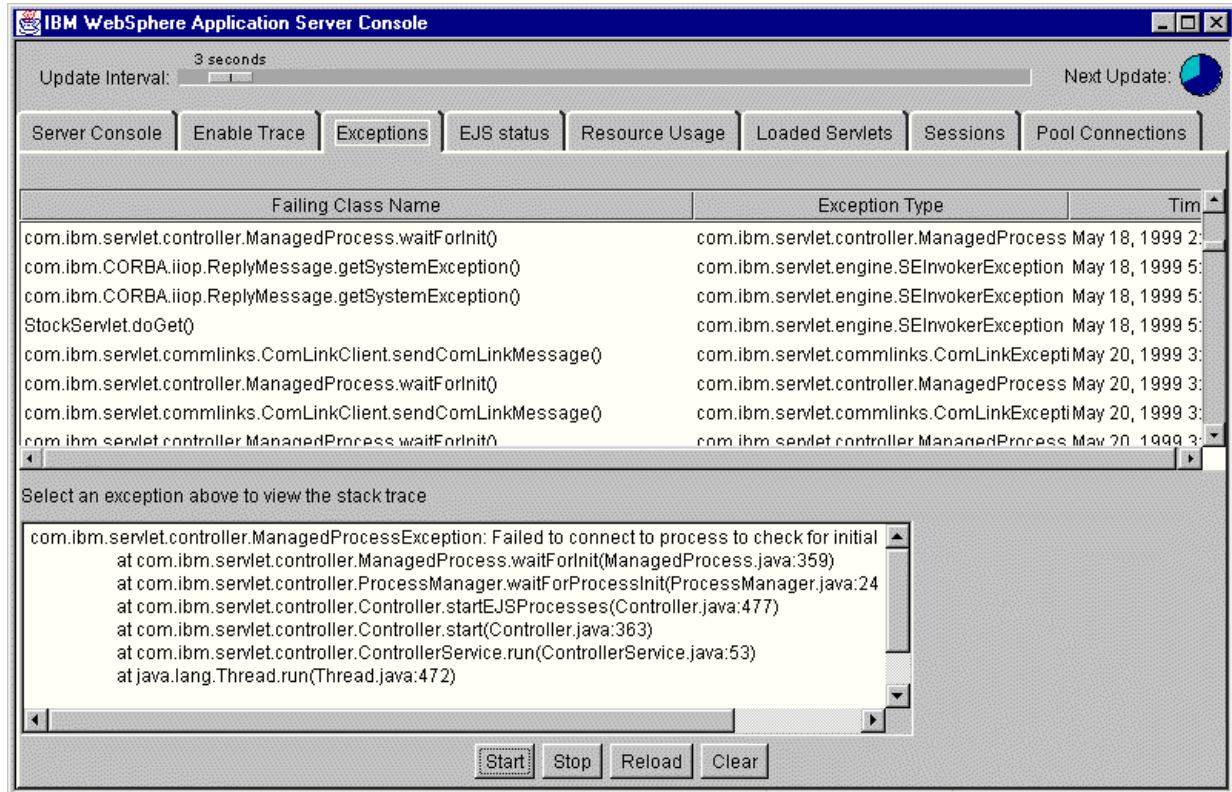


Figure 388. The Debug Console Exceptions Monitor Page

Each line in the upper box shows one exception that occurred giving the failing class name, type of exception and the timestamp when the exception occurred. To see the stack trace for a given exception, select it in the upper box. The lower box displays the stack trace for the given exception.

The exception display is not dynamic by default. To update the display either click the **Reload** button to refresh the list once or the **Start** button to automatically update the display at an interval specified by the Specify Interval slider at the top of the screen. Once automatic update has been selected, it can be stopped by clicking the **Stop** button. The time to the next update is shown in the small pie chart at the top right part of the screen.

Automatic update carries across the other monitors so that once it has been enabled for one monitor page, it continues to update all other monitor pages even when the current page is not showing. The stop button on any page will stop the update for all pages.

Select the **Clear** button to remove all current exceptions from the display.

8.2.5 The EJS Status Monitor

The EJS status monitor provides a basic check on the performance and status of the three Java processes that make up WebSphere Enterprise Java Services (see Figure 389). If any of these processes are down, then it may be worth enabling one or both of the process-specific tracers using the debug.properties file (see 8.3.4, “Setting Trace Properties Using the Debug.properties File” on page 428) and attempting to restart the server to see what messages are produced. The relevant tracer names are prefixed with LSD_ for the Location Service Daemon, PNS_ for the Persistent Name Service and EJS_ for the EJS runtime. Each process has two tracers, one for standard output suffixed with *stdout* and one for standard error suffixed with *stderr*.

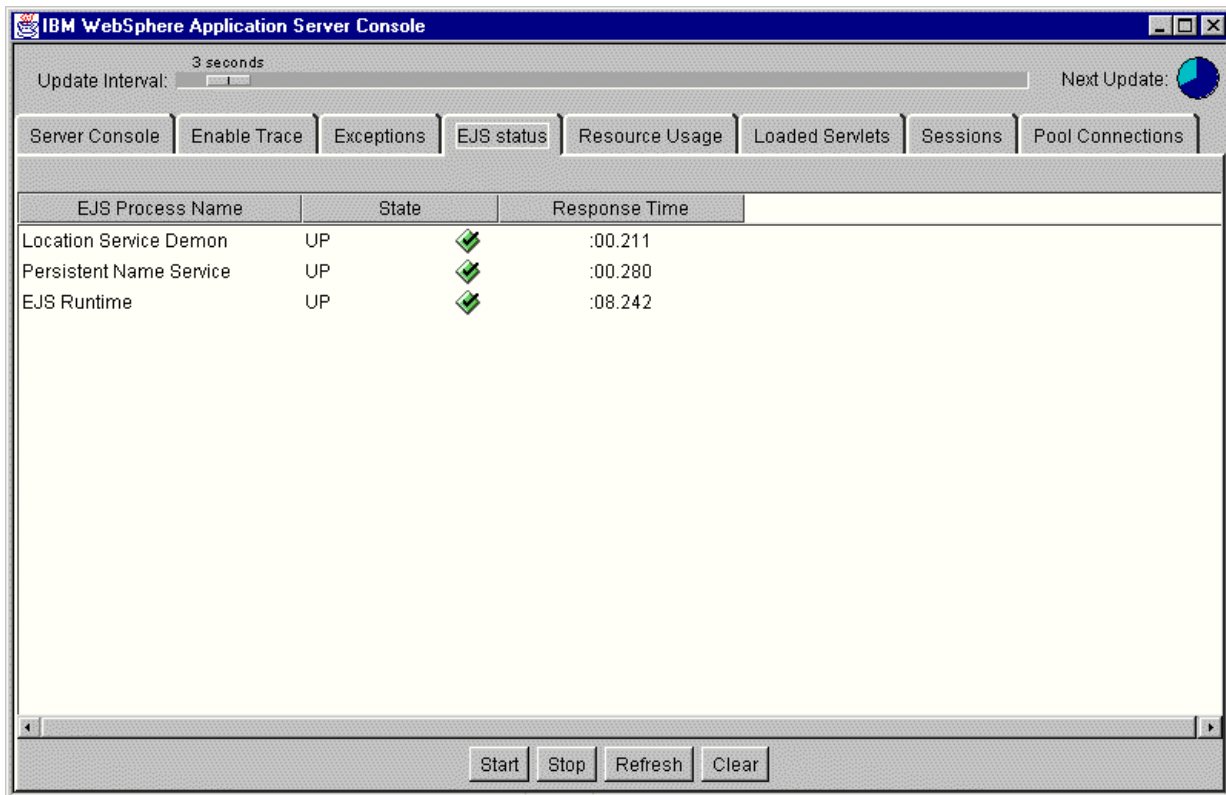


Figure 389. The Debug Console EJS Status Monitor Page

8.2.6 The Resource Usage Monitor

This monitor should be used as a basic performance monitoring tool instead of a debugging aid. It lets you see what load the server is processing at any given time. You may be able to detect memory leaks by watching the memory graph over time.

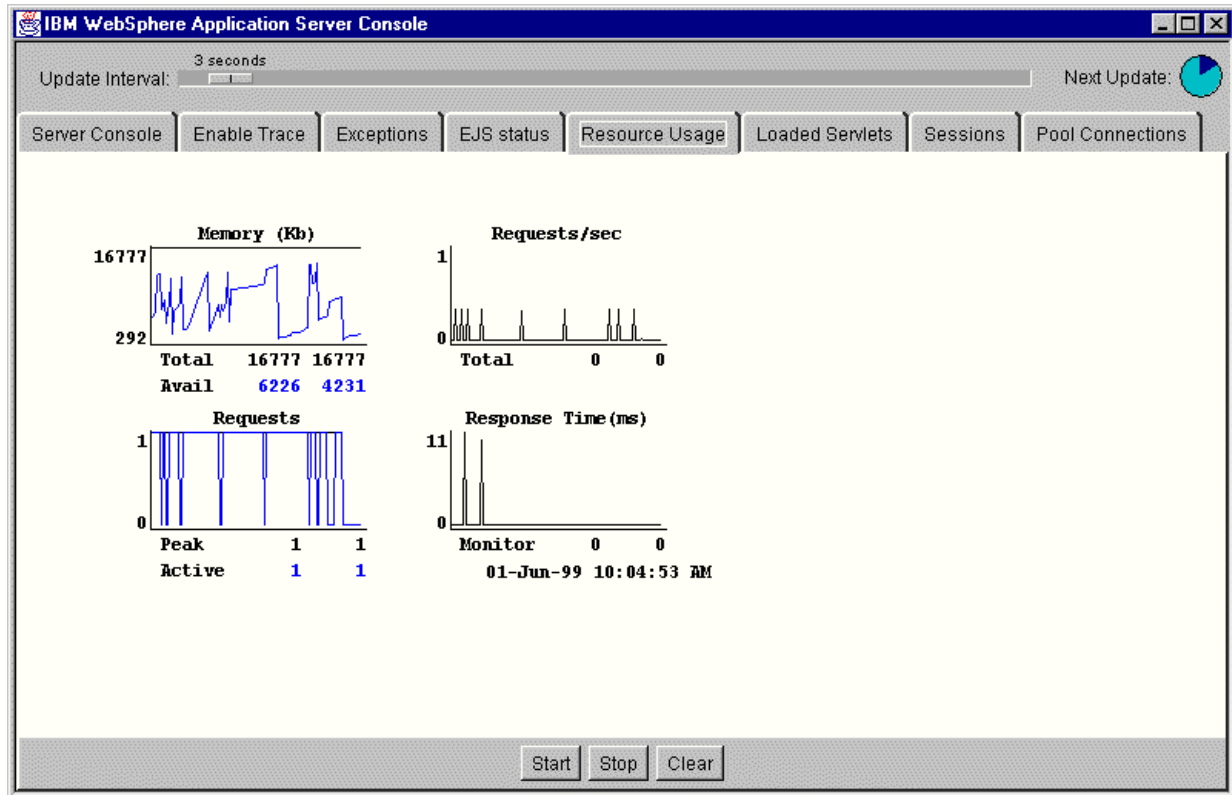


Figure 390. The Debug Console Resource Usage Monitor Page

8.2.7 The Loaded Servlets Monitor

The loaded servlets monitor allows you to see which servlets are loaded and what they are doing at any given time. Figure 391 on page 424 shows this page and a number of details for three loaded servlets. The fields and operations on this page are identical to those described in 3.1.4, “Monitoring Servlets” on page 116.

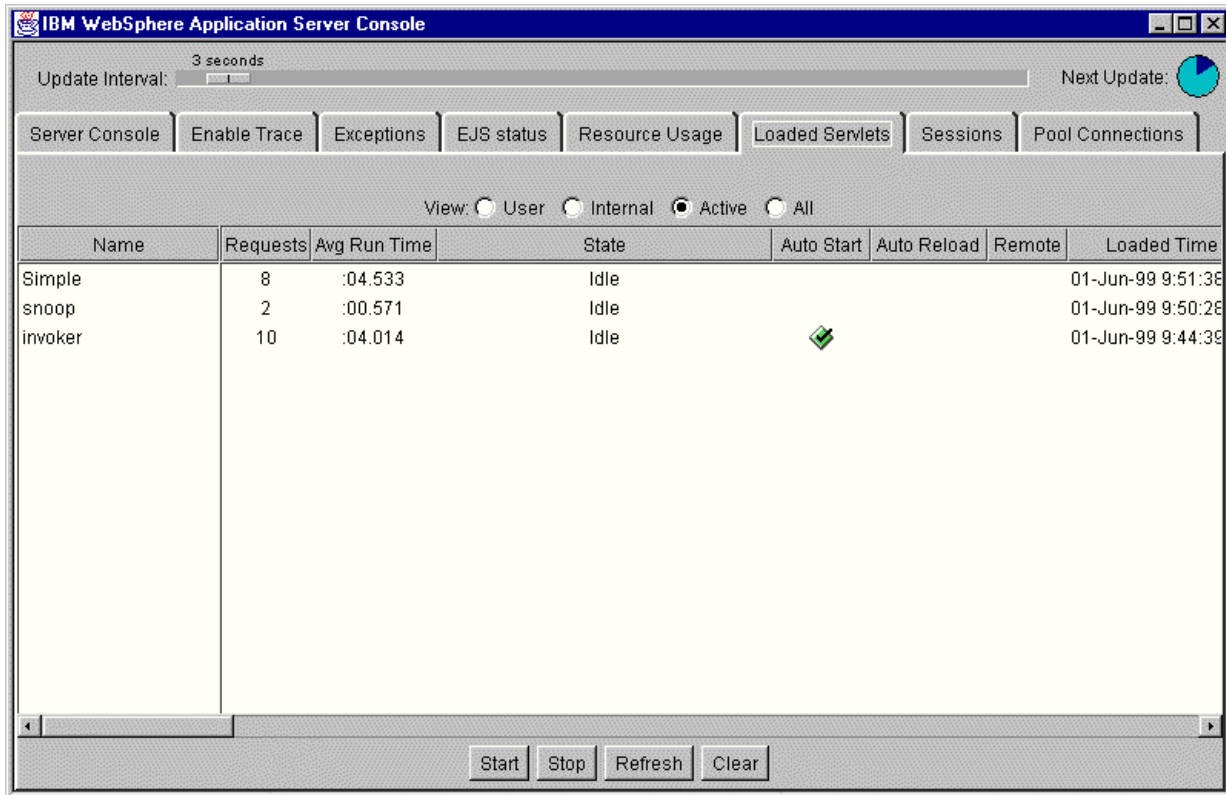


Figure 391. The Debug Console Loaded Servlet Monitor Page

8.2.8 The Sessions Monitor

The sessions monitor page provides the same information that is provided by the WebSphere administration interface Server Execution Analysis tools. See 5.1.2.2, “Monitoring HttpSession/IBMSessionData Object” on page 225 for an explanation of the information contained on this page.

8.2.9 The Pooled Connections Monitor

The pooled connections monitor page provides the same information that is provided by using the WebSphere administration interface Server Execution Analysis tools. See 5.4.2.1, “Monitoring the Connection Manager” on page 308 for an explanation of the information contained on this page.

8.3 Tracing

The WebSphere trace subsystem is based on the idea of tracers and trace output handlers. WebSphere defines a number of tracers that trace events for certain processes in the WebSphere system. Each tracer can then be associated with one or more trace output handlers that receive the tracer events and record or display them.

WebSphere also allows you to define your own tracers and tracer output handlers and link them to the WebSphere tracing subsystem. See 8.3.5, “Creating Your Own Tracers” on page 430 and 8.3.6, “Creating Your Own Trace Output Handlers” on page 431 for further details.

8.3.1 Tracers

Table 39 lists the standard tracers that appear by default in the GUI list of tracers provided by the WebSphere tracing subsystem:

Table 39. Default Tracers Available in WebSphere

Tracer Name	Message Content
ActiveDBConns	Messages from the database connection pooling subsystem.
ActiveSessions	Messages from the session manager relating to active sessions.
ClassLoader	Messages from the dynamic class loader for servlets.
Controller	Messages from the EJS managing process.
DebugConsole	Messages relating to the debug console GUI.
DebugSupport	Messages relating to the activation and deactivation of debug support.
EJS_stderr	Standard error output from the EJS container process.
EJS_stdout	Standard out output from the EJS container process.
EnableTrace	Messages relating to the Enable Trace page of the debug console.
IBMSessionContextImpl	Messages from the session manager.
LSD_stderr	Standard error messages from the EJS location service daemon process.
LSD_stdout	Standard output messages from the EJS location service daemon.
LoadedServlets	Messages relating to the loaded servlets page of the debug console.
MonEJSProcs	Messages relating to the EJS Status page of the debug console.
MonExceptions	Messages relating to the Monitor Exceptions page of the debug console.
PNS_stderr	Standard error output from the EJS persistent naming service.
PNS_stdout	Standard output from the EJS persistent naming service.
ResourceUsage	Messages relating to the resource usage page of the debug console.
SEEngine	Request response data messages from the servlet engine.
ServiceMonitoringAdmin	Messages relating to the Server Execution Analysis portion of the WebSphere Administration GUI.

Tracer Name	Message Content
adminservice.servlets	Messages relating to the life cycle of servlets used in the servlet administration GUI.
httpservice.servlets	This tracer seldom produces any output.
servletservice.servlets	Messages relating to the servlet lifecycle of user servlets.

The tracers shown in Table 39 are generally available in the GUI interface at WebSphere startup time. In addition to these there are a number of other tracers that may appear in the list depending upon whether or not the code that registers them has been executed. Table 40 lists these tracers:

Table 40. Other Tracers Provided by WebSphere

Tracer Name	Message Content
NativeServerEntry	These two tracers trace the low level connections between the C native code part of WebSphere and the Java engine.
NativeServerConnection	
AppServerConnection	These tracers are <i>all</i> associated with the IBM connection manager. If you need to trace connection manager interactions, we suggest that you turn all of these on.
IBMConnMgrServlet	
IBMConnMgr	
IBMConnPool	
IBMConnection	
IBMConnPoolSpec	
IBMConnSpec	
IBMJdbcConn	
IBMJdbcConnPool	
IBMJdbcConnSpec	
IBMConnMgrUtil	
IBMDirMgrAdmin	

A rough guide to the usefulness of a particular tracer is given by whether or not its name appears in the groups defined in the debug.properties file. If a tracer name appears, then it will probably provide useful output.

Each tracer produces messages that may consist of up to six components:

1. A timestamp
2. The tracer name
3. The level of the message
4. The message
5. The thread ID of the thread creating the message
6. Any exception thrown that relates to the event

The particular components for each message are determined by the format property of the trace output handler used. See 8.3.4, “Setting Trace Properties Using the Debug.properties File” on page 428 for more information.

Each message also has a level from 1 to 4. The levels are meant to reflect increasing levels of detail to allow filtering of the messages. The output handlers display messages less than or equal to their level Threshold parameter (see 8.3.4, “Setting Trace Properties Using the Debug.properties File” on page 428 for more information). This means that messages that are generated for level 1 have fewer details than ones that are set to level 4, and that there will be fewer level 1 messages generated. Another way of looking at this is that level 1 messages represent more serious events that all trace output handlers will want to display, whereas level 4 messages represent informational events that may or may not be of interest.

8.3.2 Trace Output Handlers

Table 41 shows the three trace output handlers provided with WebSphere:

Table 41. Trace Output Handlers Provided With WebSphere

Trace Output Handler Name	Description
LogFile	Output handler that writes trace information to a file specified in the debug.properties file.
Console	Output handler that writes trace output to the console page of the debug console.
SocketServer	Output handler that writes output to a socket server that may be running on a remote machine.

8.3.3 Running the Socket Server Trace Console

In order to see the output from the SocketServer trace output handler you need to run the socket server trace console on the machine that you wish to view or log the trace output on. To accomplish this you need to run the Java class `com.ibm.servlet.debug.SETraceServer` and perform the following steps:

1. The `com.ibm.servlet.debug.SETraceServer` class can be found in the `<Server Root>\lib\ibmwebas.jar` file. You need to copy this file to the machine you wish to trace from and make sure that the JAR file is included in the java classpath from that system.
2. Set the `trace.handler.SocketServer.param.server` and `trace.handler.SocketServer.param.port` parameters in the `debug.properties` file to the address of the tracing machine and the port number you want to use for tracing on the tracing machine respectively. The port number defaults to 9991 on the server, so use this value unless the tracing machine already has something running on that port. See 8.3.4, “Setting Trace Properties Using the Debug.properties File” on page 428 for more information on how to accomplish this.
3. Stop and restart WebSphere.
4. On the tracing machine (as opposed to the traced machine) execute the `SETraceServer.class` in the Java environment on that machine. On machines with command lines, this can be accomplished by typing the following at a command prompt:

```
java com.ibm.servlet.debug.SETraceServer <port number> <file name>
```

where `<port number>` is the number of the port the server will listen to receive trace messages that you specified in the `trace.handler.SocketServer.param.port` parameter in step 2, and `<file name>` is the name of a file to write trace output to in addition to writing it to the console. Both of these parameters are optional. If they are omitted, then the port number defaults to 9991 and no file output is produced. On machines without a command line (such as the Apple Macintosh), use the functions on that platform to set these command line parameters.

You should see messages like those shown in Figure 392, depending upon the parameters you specified:

```
Starting the WebSphere Trace Server.  
Creating server socket on port 8888.  
Sending output to the screen and to the file d:\temp\trace.log.
```

Figure 392. Messages Output by the Socket Server

5. You should now be able to enable tracer output to the `SocketServer` and have it appear as a message on the socket server console as well as logged to the file that you specified in step 4.

8.3.4 Setting Trace Properties Using the `Debug.properties` File

The `<Server Root>/properties/server/servlet/debug.properties` file is the file that WebSphere reads at startup to determine the configuration of the trace and debug subsystems. Some of the trace subsystem options can only be set in this file, while others take their defaults from here.

The `debug.properties` file allows more granular control over tracers by allowing them to be assigned to trace groups. Each group can be turned on or off and assigned one or more output handlers to receive output from the group. Tracers can also be manipulated on a global basis by using the `traceAll` settings.

Table 42 lists the possible trace properties in the `debug.properties` file:

Table 42. Trace Properties Available in the `debug.properties` File

Property Name	Description
<code>trace.traceAll.state</code>	Set to on to enable all tracers by default.
<code>trace.traceAll.handlers</code>	Space separated list of trace output handlers to receive trace events when global tracing is turned on.
<code>trace.groups</code>	Space separated list of trace group.
<code>trace.group.<group name>.tracers</code>	Space separated list of tracers to be included in the tracer group <code><group name></code> .
<code>trace.group.<group name>.state</code>	Set to on to enable all tracers for the group <code><group name></code> .
<code>trace.group.<group name>.handlers</code>	Space separated list of trace output handlers to receive trace events when tracing for the group <code><group name></code> is turned on.
<code>trace.tracers</code>	Space separated list of individual tracer names.

Property Name	Description
trace.group.<tracer name>.state	Set to on to enable tracing for the tracer <tracer name>.
trace.group.<tracer name>.handlers	Space separated list of trace output handlers to receive trace events when tracing for the tracer <tracer name> is turned on.
trace.handlers	Space separated list of trace output handlers.
trace.handler.<handler name>.class	Name of the Java class providing the handler functions for output handler <handler name>.
trace.handler.<handler name>.params	Space separated list of parameters that handler <handler name> accepts.
trace.handler.<handler name>.param.<parameter name>	Value for the parameter <parameter name> for the handler <handler name>.
trace.handler.<handler name>.param.format	Integer value from 0 to 4 specifying which items are to be placed in the trace message.
trace.handler.<handler name>.param.levelThreshold	Integer value from 1 to 4 which specifies the maximum level of messages to be displayed by handler <handler name>.
trace.handler.LogFile.param.filename	Name of the file that the LogFile output handler uses to place trace output in.
trace.handler.SocketServer.param.server	Name or IP address of the host running the socket server for the SocketServer output handler.
trace.handler.SocketServer.param.port	Port address for the socket server on the host running the socket server for the SocketServer output handler.

Notes:

1. Tracer names, group names, handler names and parameter names must all contain no internal spaces, since the lists of these names are space delimited. Use underscores instead.
2. The format setting for output handlers specifies the message format as shown in Table 43:

Table 43. Trace Output Handler Format Settings

Format Setting	Resulting Message Components
0	Displays Timestamp, ThreadID, Tracer Name, Message, Exception
1	Displays Timestamp, Tracer Name, Message, Exception
2	Displays Timestamp, Tracer Name, Message
3	Displays Tracer Name, Message
4	Displays Just the Message

User-defined output handlers may ignore or change these formats.

- The levelThreshold setting specifies the maximum level of trace messages that are displayed by the output handler. If the threshold is 3, then only messages of levels 1, 2 and 3 will be seen. User-created output handlers may or may not respect this setting.

8.3.5 Creating Your Own Tracers

Note: This API *will* change in Version 3.0 of WebSphere. Code that you create using these instructions will have to be rewritten to work correctly with Version 3.0. There is a new IBM strategic way of handling tracing that will be included in WebSphere Version 3.0.

If the tracers available under WebSphere do not suit the needs of your application debugging, it is possible to create your own tracers to produce debugging output using the WebSphere tracing infrastructure. To create user-defined tracers in V2.02 perform the following steps:

- In your application code declare a variable of type `com.ibm.servlet.debug.SETracer`. This type can be found in the file `<Server Root>\lib\ibmwebas.jar` and in the IBM VisualAge for Java WebSphere test environment. It makes sense to declare the variable as static unless you want different tracer names for different instances of your class. If the class is a servlet then by declaring the variable static the tracer will be available as soon as the servlet is loaded.
- Initialize the variable by calling the `SETracer` constructor with two string arguments. The first argument is the name of the tracer that will appear in the trace output. This name must not contain any spaces. The second argument is the description of the tracer. For example, the following line both declares and initializes a static tracer instance called `testTracer`:

```
private static com.ibm.servlet.debug.SETracer testTracer = new
com.ibm.servlet.debug.SETracer("My_Tracer", "My Test Tracer");
```

If the variable is declared as static, put this code in the static section of your class outside the methods.

- In your code you can then call one of the trace methods on `SETracer` to write a trace message. The following is a list of all of the different methods:

```
public void trace(String msg)
public void trace(String msg, int level)
public void trace(String msg, Exception exc)
public void trace(String msg, Exception exc, int level)
```

Table 44 lists the possible parameters. Parameters omitted from the parameter list on a particular trace method invocation take the default value.

Table 44. Possible Parameters for the `SETracer` trace Method

Parameter Name	Description	Default Value
msg	The message that is to be associated with this tracing event.	N/A
level	The level of this message. Level 4 messages should represent more detail than level 1 messages.	1
exc	The exception associated with this event.	null

4. When running your code you will need to register the tracer with the tracer manager first before the tracer becomes visible on either the debug console Enable Trace page or the Trace page of the WebSphere administration interface. To do this, you must execute the code that calls the tracer constructor. If the code to perform the initialization is static and your class is a servlet, loading the servlet will achieve this. Otherwise, you may need to execute the code once to register the tracer with the tracer manager before it is visible in the interface.

Note: Once a tracer is registered with the tracer manager, you will not be able to register a new instance of the tracer class with the same name and have it work correctly. This is why it is better to have the tracer registration occur in static code. If you need to have the piece of code that registers the tracer executed a second time you have to stop and restart WebSphere before tracing will work again. This is particularly relevant to servlet reloading.

5. After the tracer has been constructed, you should be able to click the **Refresh** button on the tracer page of either the debug console or the WebSphere administration interface and see your tracer added to the list. You can then manipulate it as you would other tracers, with one important exception. As the tracer class is not loaded at WebSphere startup time, it will not be available for loading through the use of the debug.properties file.

8.3.6 Creating Your Own Trace Output Handlers

Note: This API *will* change in Version 3.0 of WebSphere. Code that you create using these instructions will have to be rewritten to work correctly with Version 3.0. There is a new IBM strategic way of handling tracing that will be included in WebSphere Version 3.0.

The WebSphere trace utilities makes use of three classes to provide output handlers for the trace messages. These three handlers provide output to the console, to a log file and to a socket server. If the behavior of these handlers is not to your liking, it is possible to create user-defined output handlers to process trace messages.

Figure 393 on page 432 shows the code for a simple trace output handler.

```

import java.io.*;
import java.util.*;
import com.ibm.servlet.debug.*;

public class TestOutputHandler extends SETracerOutputHandler
{
    private String myParameter;
    public TestOutputHandler()
    {
        super();
    }
    /**
     * This method was created in VisualAge.
     * @return java.lang.String
     */
    public String getMyParameter() {
        return myParameter;
    }

    public void init(Hashtable params) {
        super.init(params);
        setMyParameter((String)params.get("myParameter"));
    }

    /**
     * This method was created in VisualAge.
     * @param newValue java.lang.String
     */
    public void setMyParameter(String newValue) {
        this.myParameter = newValue;
    }

    /**
     * This method was created in VisualAge.
     * @param timestamp java.lang.String
     * @param threadID java.lang.String
     * @param tracerName java.lang.String
     * @param message java.lang.String
     * @param exc java.lang.Throwable
     * @param severity int
     */
    public void write(String timestamp, String threadID, String tracerName,
String message, Throwable exc, int severity) {

        System.out.println(getMyParameter() + "\t" + timestamp + "\t" +
tracerName + "\t" + severity + "\t" + threadID + "\t" + message);
        if (exc != null) {
            System.out.println("exc: ");
            exc.printStackTrace(System.out);
        }
    }
}

```

Figure 393. A Simple Trace Output Handler

The following points are worth noting:

1. The class inherits from `com.ibm.servlet.debugSETracerOutputHandler`, which is the base class for WebSphere output handlers. This class is contained in the `<Server Root>\lib\ibmwebas.jar` file and also in the VisualAge for Java WebSphere test environment. This is the type that WebSphere expects in its list of output handlers so user-defined output handlers need to extend this class.
2. The class declares a private String variable `myParameter` in order to illustrate how tracer configuration information can be passed to a handler. Getter and setter methods are defined for this variable. In a more complex output handler there may be a number of parameters specified.
3. The class overrides the `init(Hashtable)` method to capture the new parameter value from the hashtable of parameters passed to it by WebSphere. It also calls the superclass `init` method to take care of the standard parameters. The superclass `init` method should always be called by a subclass that overrides it. Step 2 on page 434 discusses how to specify user-defined parameter values in the `debug.properties` file so that they get included in the input hashtable.
4. The main method that needs to be overridden is the `write` method. This is the method that is called by WebSphere to notify the output handler that a trace event has occurred. The parameters for this method give information about the trace event:

Table 45. Parameters to the Write Method for a Trace Output Handler

Name	Description	Example
<code>timeStamp</code>	The date and time that the trace event occurred.	6/2/99 11:42:58 AM EDT
<code>threadID</code>	The thread identifier of the thread that caused the event.	Thread[main,5,main]
<code>tracerName</code>	The name of the WebSphere tracer that generated the event.	SEEngine
<code>message</code>	The actual message generated by the tracer.	SEEngine: created successfully
<code>exc</code>	Any exception generated by this event. This value may be null as not all events generate exceptions.	N/A
<code>severity</code>	The level of this event. Event levels can range from 1 to 4.	1

5. In the `write` method no attempt is made to handle the filtering and format options specified in the input parameters. All messages are displayed and all items are printed. If you need to get access to these parameters you can call the `getLevelThreshold()` method which returns the int level threshold setting and the `getFormat()` method which returns the int format setting. See the comments in the `debug.properties` file for a description of what these numbers are supposed to mean.
6. In the `write` method of this handler the trace information is simply written to standard output, which means that it ends up in the standard output log as well as appearing on the debug console if enabled. It is possible to redirect the output to a remote socket server or to a systems management console, or to process the output in other ways from this method. Note that the exception information is only printed if it is not null.

To enable the output handler it needs to be defined in <Server Root>\properties\server\servlet\debug.properties. To add an output handler to debug.properties perform the following steps:

1. Find the definition for trace.handlers near the bottom of the file and add the name of the new output handler to the end. For example, to add a new output handler called Test, the line would look like this:

```
# Enabled Output Handlers
trace.handlers=Console LogFile SocketServer Test
```

Note that the handler name should not contain any spaces.

2. Create a properties section for the new handler that looks something like the following:

```
# Test Handler Properties
trace.handler.Test.class=TestOutputHandler
trace.handler.Test.params=myParameter format levelThreshold
trace.handler.Test.param.myParameter=Hello World
trace.handler.Test.param.format=2
trace.handler.Test.param.levelThreshold=4
```

Each property must be prefixed with trace.handler.<your handler name>. where <your handler name> is the name of the handler that you defined in step 1. The class parameter specifies the class name of the handler, which must be in the application server classpath. The params parameter specifies the names of the other parameters that WebSphere will look for to initialize your class. These parameter names and the corresponding values will be placed into a hashtable that will be passed to the init method of your output handler code. The levelThreshold and format parameters must be present (see the comments in the debug.properties file for the definitions of these parameter values). User-defined parameter values can be added as needed. Notice in this example that the myParameter value used in the sample code in Figure 393 on page 432 is given the value "Hello World".

3. Stop and restart the WebSphere servlet service to make these changes active.

After completing these steps you should be able to see the new trace output handler in the Enable Trace page on the debug console and on the Trace page in the Server Execution Analysis page of the WebSphere administration interface. Figure 394 on page 435 shows the Enable Trace page from the debug console showing the new Test output handler column.

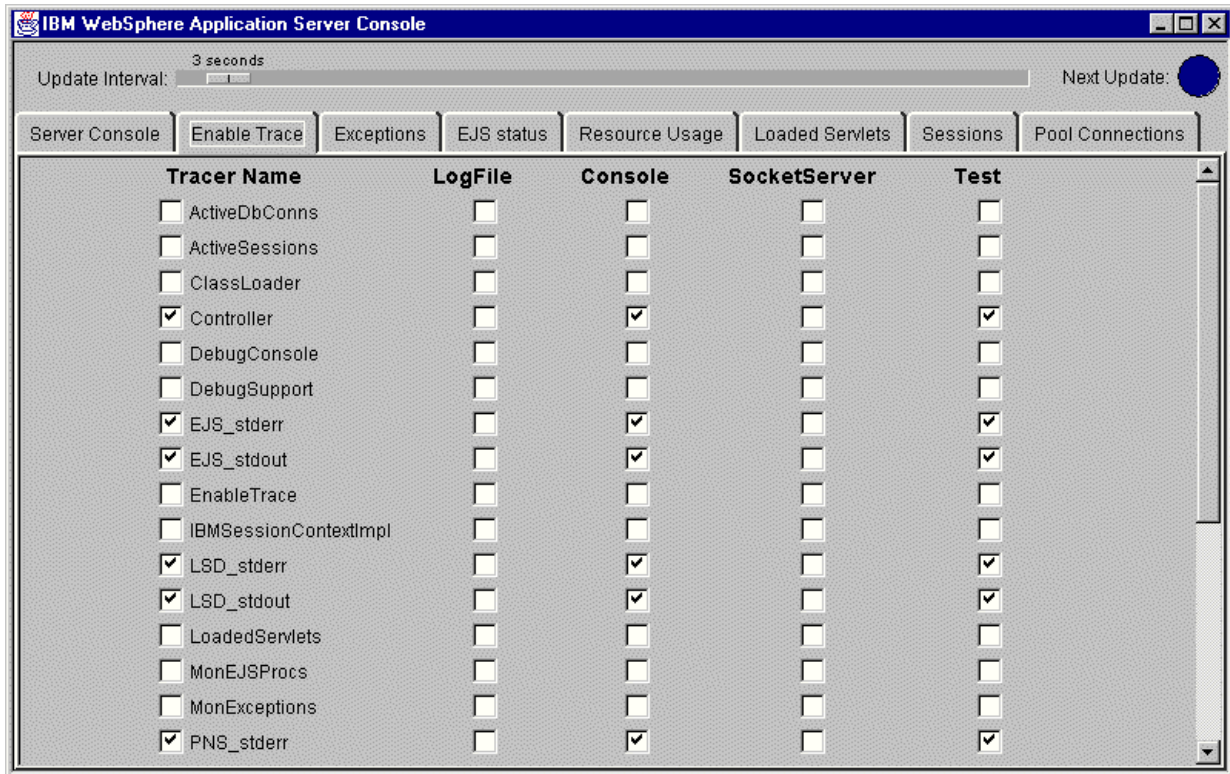


Figure 394. The Debug Console Showing a Custom Output Handler

8.4 The Server Execution Analysis Pages

The WebSphere administration GUI includes a number of pages dedicated to the monitoring and debugging of the WebSphere system. These pages utilize the same Java applets that the debug console utilizes and thus work the same way. Table 46 lists the Server Execution Analysis pages that have corresponding debug console pages and cross references to the sections that discuss them:

Table 46. Server Execution Analysis

Server Execution Analysis Page Name	Debug Console Page Name	References to Book Sections
Active Sessions	Sessions	8.2.8, "The Sessions Monitor" on page 424.
DB Pool Connections	Pool Connections	8.2.9, "The Pooled Connections Monitor" on page 424.
EJS Status	EJS Status	8.2.5, "The EJS Status Monitor" on page 422.
Exceptions	Exceptions	8.2.4, "The Exceptions Monitor" on page 421.
Loaded Servlets	Loaded Servlets	8.2.7, "The Loaded Servlets Monitor" on page 423.

Server Execution Analysis Page Name	Debug Console Page Name	References to Book Sections
Resource Usage	Resource Usage (The Server Execution Analysis Page adds a tabular view of the data not found in the debug console)	8.2.6, "The Resource Usage Monitor" on page 422.
Trace	Enable Trace	8.2.3, "The Trace Enabler Page" on page 420.

The following sections describe the other pages in the Server Execution Analysis pages.

8.4.1 The JVM Debug Page

The JVM debug page controls the settings relating to attaching a remote debugger, such as the jdb debugger provided with the JDK, to the main WebSphere JVM. This allows you to run WebSphere as if you had typed `java -debug` to run the WebSphere main JVM. For more details, see:

<http://java.sun.com/products/jdk/1.1/docs/tooldocs/win32/index.html>

8.4.2 The Settings Pane

The settings pane (See Figure 395 on page 437) allows you to control the settings for JVM remote debugging. For details on the fields on this page, see:

<http://<Your Server Name>:9527/admin/webexec/WASHelp/monitor.htm#s>

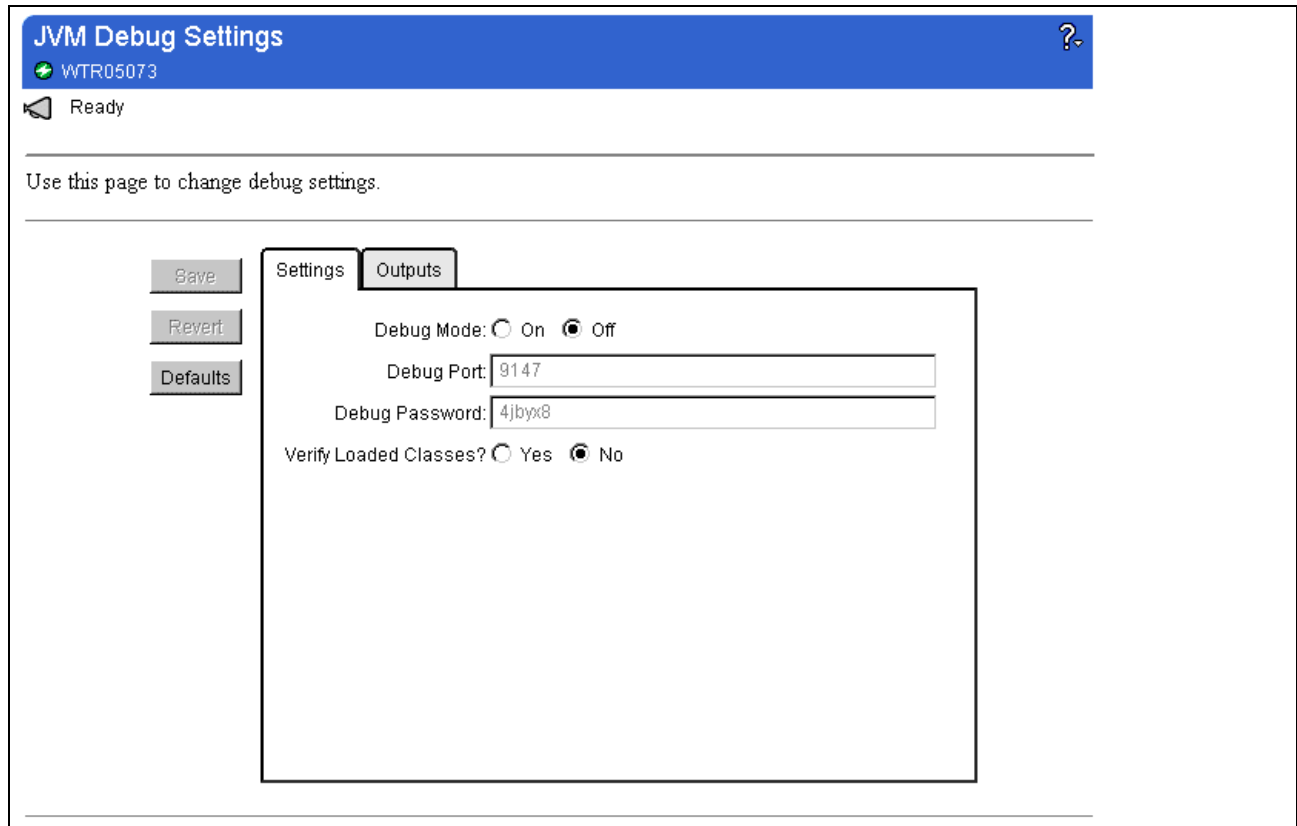


Figure 395. The JVM Debug Settings Pane

8.4.2.1 The Output Pane

The output pane (see Figure 396 on page 438) allows you to control the contents of the debug output from the WebSphere JVM as well as allowing you to enable the debug console. For details on the fields on this page, see

<http://<Your Server Name>:9527/admin/webexec/WASHelp/monitor.htm#o>

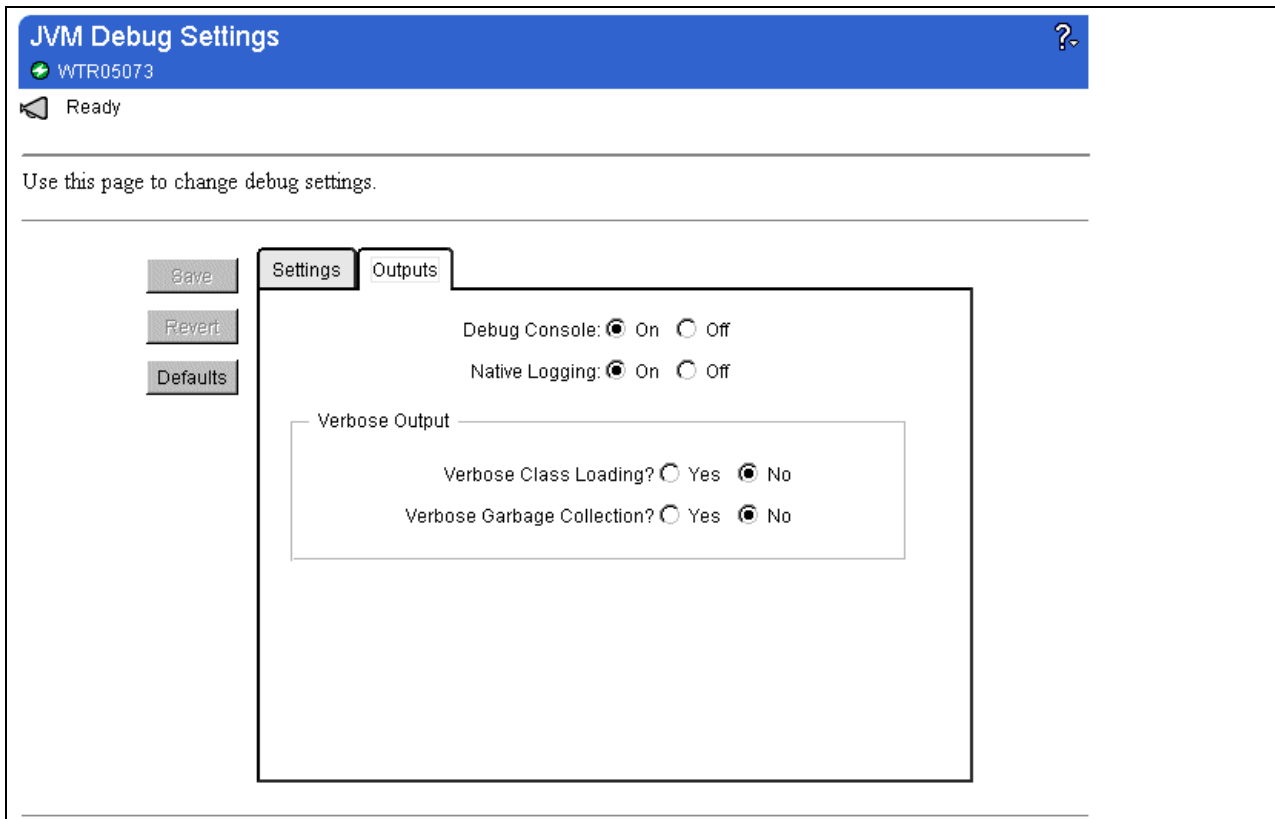


Figure 396. The JVM Debug Settings Output Pane

8.4.3 The Error Log Settings Page

The error log settings page (see Figure 397 on page 439) allows you to configure the settings and disposition of the WebSphere servlet error logs (see 8.1.10, “The Servlet Service Error Log” on page 417). The fields on this page are described at:

<http://<Your Server Name>:9527/admin/webexec/WASHelp/monitor.htm#fe>.

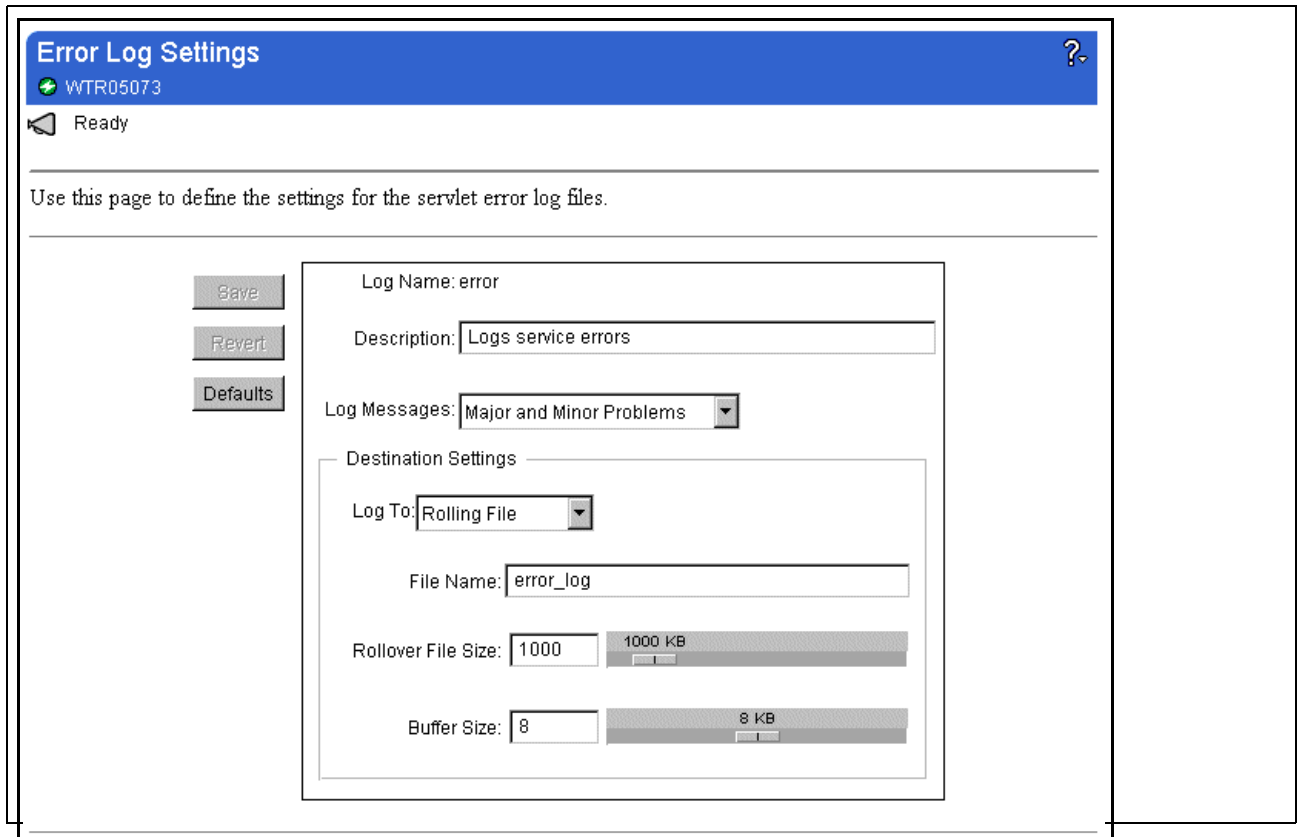


Figure 397. The Error Log Page

8.4.4 The Event Log Settings Page

The event log settings page (see Figure 398 on page 440) allows you to configure the settings and disposition of the WebSphere servlet event logs (see 8.1.11, “The Servlet Service Event Log” on page 418). The fields on this page are described at:

<http://<Your Server Name>:9527/admin/webexec/WASHelp/monitor.htm#fv>.

Figure 398. The Event Log Settings Page

8.4.5 The Dump Panel Setup Page

The dump panel setup page allows you to create a snapshot of a WebSphere system either immediately or automatically when certain conditions are met. The automatic settings allow dumps to be taken if a servlet hangs or if JVM memory reaches a certain threshold. This can be useful in helping to diagnose problems where a servlet or JSP hangs intermittently or has a memory leak.

The dump, which can be logged to a file, includes the following information:

- A snapshot of the server properties in effect when the dump was taken
- A dump of the information for every thread executing in the WebSphere system
- A snapshot of the state of the servlet runtime including a list of loaded servlets
- A snapshot of any database connections.
- A snapshot of recent exceptions
- Other WebSphere state data

Figure 399 on page 441 shows the Dump Panel Setup page. For more information on the fields on this page, see:

<http://<Your Server Name>:9527/admin/webexec/WASHelp/monitor.htm#mondu4>.

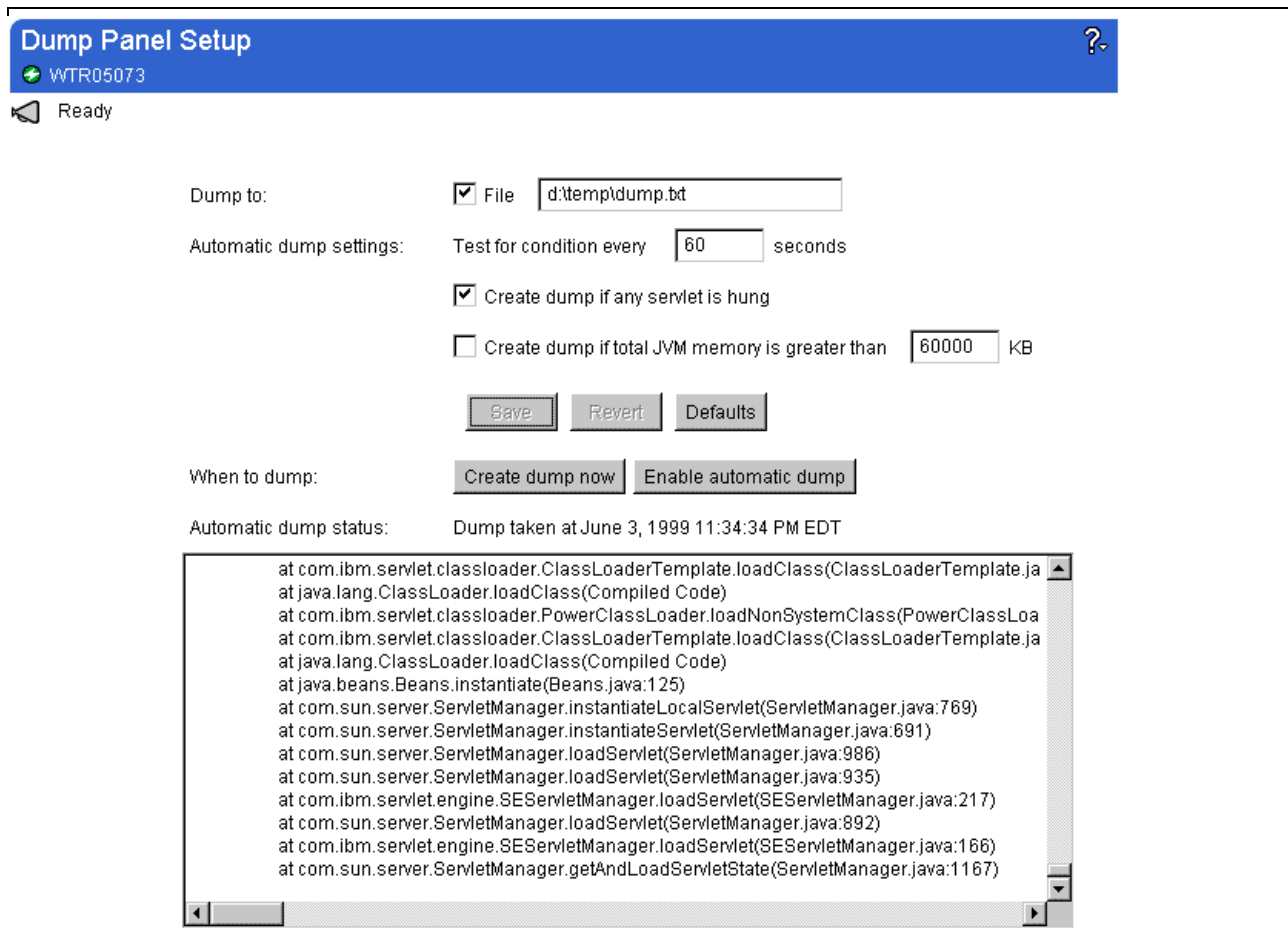


Figure 399. The Dump Panel Setup Page

8.4.6 The Log Output Monitor Page

The log output monitor page provides you with another view of either the JVM standard error or standard output logs. This page allows you to view a snapshot of the end of either the standard error or the standard output logs. The amount of the log that you see is configurable. The snapshot can be set for automatic update.

This page is useful when you want to monitor the logs on an ongoing basis without necessarily looking at all of them.

Figure 400 on page 442 shows a section of the log output monitor page. For more information on the fields on this page see:

<http://<Your Server Name>:9527/admin/webexec/WASHelp/monitor.htm#flo>

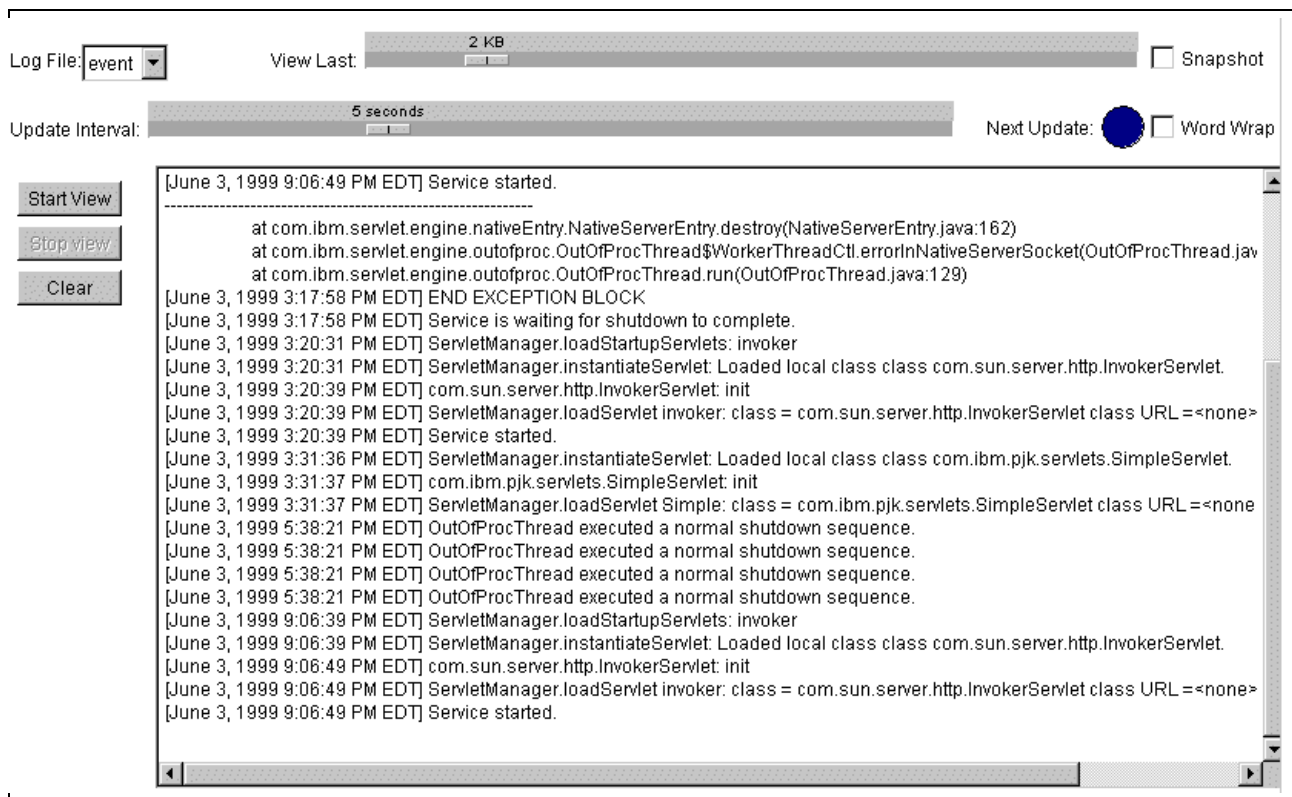


Figure 400. The Log Output Monitor Page

8.5 Miscellaneous Debugging Tools

There are a few other non-WebSphere centric tools that can be used to assist in debugging your environment.

8.5.1 DB2 CLI Tracing

One method for determining what is going on at a low level with DB2 is to use the DB2 CLI Trace facility. This will allow you to see exactly what calls the JDBC driver is or is not making to the database. This technique is more useful in debugging problems loading DB2 drivers rather than problems with application logic, since if the JDBC driver is talking to the database successfully the problems are generally at a higher level. This technique is also useful if detailed debugging information is required for EJB persistence problems.

To enable DB2 CLI tracing do the following:

1. Edit the file db2cli.ini (it is possible to perform this task using the GUI but the GUI differs between platforms)
2. Look for a line similar to the following:

```
[common]
```

If it does not exist add it to the bottom of the file.

3. Add the following lines after the [common] line:

```
TRACEFLUSH=1
TRACEFILENAME=<trace file name>
TRACE=1
```

4. If one does not already exist, add a section for each database you want to trace using the following template as a guide:

```
[<database name>]
PWD=<password>
UID=<userid>
APPENDAPINAME=1
DBALIAS=<ODBC name>
```

Where the <ODBC name> should probably be the same as the <database name>.

The APPENDAPINAME parameter adds the name of the API being called to the trace output which gives a much better idea of what is going on.

5. Restart DB2

Figure 401 shows an extract from an actual db2cli.ini file with two databases defined for tracing, the sample database and the EJS samples database:

```
[EJS_SAMP]
PWD=swa109r
UID=peterk
APPENDAPINAME=1
DBALIAS=EJS_SAMP

[Common]
TRACEFLUSH=1
TRACEFILENAME=d:\traces\cli\trace.log
TRACE=1

[SAMPLE]
PWD=swa109r
UID=peterk
APPENDAPINAME=1
DBALIAS=SAMPLE
```

Figure 401. Extract From DB2CLI.INI Showing Tracing Parameters

Figure 402 on page 444 is an extract from a DB2 CLI trace log showing the allocation of the DB2 environment and an initial connection to a database that fails due to a missing user ID and password. Typically if you see the SQLDriverConnect call being executed after the previous calls have succeeded, then the driver has registered correctly.

```

SQLAllocEnv( phEnv=&682cd80 )

SQLAllocEnv( phEnv=0:1 )
<--- SQL_SUCCESS   Time elapsed - +1.000000E-002 seconds

SQLSetEnvAttr( hEnv=0:1, fAttribute=SQL_ATTR_ODBC_VERSION, vParam=3,
cbParam=0 )
---> Time elapsed - +0.000000E+000 seconds

SQLSetEnvAttr( )
<--- SQL_SUCCESS   Time elapsed - +2.000000E-002 seconds

SQLGetEnvAttr( hEnv=0:1, fAttribute=Unknown value 1271, pParam=&5eef630,
cbParamMax=0, pcbParam=NULL )
---> Time elapsed - +0.000000E+000 seconds

SQLGetEnvAttr( pParam=0 )
<--- SQL_SUCCESS   Time elapsed - +0.000000E+000 seconds

SQLAllocConnect( hEnv=0:1, phDbc=&5eef7f8 )
---> Time elapsed - +1.823000E+000 seconds

SQLAllocConnect( phDbc=0:1 )
<--- SQL_SUCCESS   Time elapsed - +0.000000E+000 seconds

SQLDriverConnect( hDbc=0:1, hwnd=0:0, szConnStrIn="DSN=sample;UID=;PWD=",
cbConnStrIn=20, szConnStrOut=NULL, cbConnStrOutMax=0, pcbConnStrOut=NULL,
fDriverCompletion=SQL_DRIVER_NOPROMPT )
---> Time elapsed - +2.400000E-001 seconds

SQLDriverConnect( )
<--- SQL_ERROR     Time elapsed - +3.075000E+000 seconds

SQLError( hEnv=0:1, hDbc=0:1, hStmt=0:0, pszSqlState=&5eef724,
pfNativeError=&5eef738, pszErrorMsg=&5eef31c, cbErrorMsgMax=1024,
pcbErrorMsg=&5eef74a )
---> Time elapsed - +2.300000E-001 seconds

SQLError( pszSqlState="42602", pfNativeError=-567, pszErrorMsg="[IBM] [CLI
Driver] SQL0567N "SYSTEM" is not a valid authorization ID.  SQLSTATE=42602

```

Figure 402. Extract From a DB2 CLI Trace Log

Note: This trace produces a lot of output. Make sure that you have enough disk space and consider turning it off after you have diagnosed the problem.

8.5.2 JDBC Output Redirection

Another useful database debugging tool is to redirect the JDBC log stream to System.out so that it can be captured in the WebSphere standard output logs. To enable this tracing put the following line in your Java code immediately before driver registration:

```
java.sql.DriverManager.setLogStream(System.out);
```


If the driver registration is successful you should see something like the output shown in Figure 403 in the standard output log:

```
DriverManager.getConnection("jdbc:db2:sample")
  trying
driver [className=COM.ibm.db2.jdbc.app.DB2Driver, context=null, COM.ibm.db2.j
dbc.app.DB2Driver@3406c2]
getConnection returning
driver [className=COM.ibm.db2.jdbc.app.DB2Driver, context=null, COM.ibm.db2.j
dbc.app.DB2Driver@3406c2]
```

Figure 403. Sample JDBC Log Output Showing a Successful DB2 Driver Connection

8.5.3 Running the EJS Processes Stand-alone

Readers familiar with the VisualAge for Java WebSphere test environment will know that the three EJS server processes can produce large quantities of output on their operation. Since it is likely that you would need a developer to help review the output, how can a code developer get access to this output in WebSphere proper? The answer is twofold.

The first way is to enable the WebSphere EJS tracers for each of the EJS processes. This has the effect of directing the output produced by these processes to the trace output destination selected. See 8.3.4, “Setting Trace Properties Using the Debug.properties File” on page 428 or 8.2.3, “The Trace Enabler Page” on page 420 for further details.

The second way is to run the EJS processes and the EJS client outside the WebSphere environment and monitor their output. To accomplish this task perform the following steps.

1. Create a script file that sets the classpath to include the following directories and files:
 - <your EJB client JAR file>
 - <Server Root>\deployedEJBs\<your EJB server JAR file>
 - <Server Root>\properties\ejb\IBMNameServiceConfig.properties
 - <Server Root>\lib\ibmwebas.jar
 - <Server Root>\lib\jst.jar
 - <Server Root>\lib\jsdk.jar
 - <Server Root>\lib\xml4j.jar
 - <Server Root>\lib\databaseans.jar
 - <Server Root>\lib\ejb.jar
 - <Database Home>\<JDBC driver files>

Call this file setclasspath and make it an executable file (use a .bat extension for Windows NT and set the appropriate execute permissions on AIX). Figure 404 on page 446 shows a sample setclasspath.bat file for Windows NT.

```

set WASL=D:\WebSphere\AppServer\lib
set DEPL=D:\WebSphere\AppServer\deployedEJBs
set CLASSPATH=%CLASSPATH%;D:\Data\MyHelloClient.jar;
set CLASSPATH=%CLASSPATH%;%DEPL%\MyHelloServer.jar;
set
CLASSPATH=%CLASSPATH%;D:\WebSphere\AppServer\properties\ejb\IBMNameService
Config.properties
set
CLASSPATH=%CLASSPATH%;%WASL%\ibmwebas.jar;%WASL%\jst.jar;%WASL%\jsdk.jar;%
WASL%\xml4j.jar
set CLASSPATH=%CLASSPATH%;%WASL%\databeans.jar;%WASL%\ejb.jar
set CLASSPATH=%CLASSPATH%;D:\sqllib\java\db2java.zip

```

Figure 404. Batch File to Set the Classpath for EJS Standalone Execution on NT

2. Create another file called startns1 (again with the appropriate attributes to make it executable on your operating system) and place the following executable command in the file all on one line:

```
java com.ibm.lsd.LocationServiceDaemon -ORBListenerPort 9029
```

3. Create another executable file called startns2 and put the following command all on one line in the file:

```
java com.ibm.CosNaming.PersistentNameServer -ORBBootstrapPort 9019
-ORBPersIORHostName wtr05073 -ORBPersIORPort 9029 -InitialRoot <Name Space
Directory>
```

Where <Name Space Directory> is an empty directory on your machine that must already exist.

4. Create a third executable file called startejs and place the following command on one line in the file:

```
java com.ibm.ejs.server.EJServer -ORBPersIORPort 9029 -ORBBootstrapPort
9019 -file <Server Root>\properties\ejb\ejb.properties
```

5. Save all of the files in the <Server Root> directory.
6. Stop the WebSphere EJS processes.

On Windows NT you can do this by stopping the Web server, stopping the WebSphere Servlet Service, performing steps 7 and 8 and then restarting the Web server.

On AIX you can simply kill the EJS processes (see Figure 165 on page 187 for details).

7. Start three command prompts, and in each of them change the directory to the <Server Root> directory and run setcpath. You may want to increase the number of lines in the command prompt windows to capture all of the output.
8. In the first command prompt window, run startns1 and wait for it to initialize. Repeat this procedure for startns2 and startejs in the second and third command prompt windows respectively.
9. Open a fourth command prompt window, run setcpath and then run your client code.

If all of the above has worked, then your code should be working properly. If any of the above steps has failed then the output from the EJS processes will hopefully give you some insight into why things are not working.

Appendix A. WebSphere Samples

1. Update the servlet files for the XtremeAdventures and WebBank samples with the DB2 user ID and password information.

To update the servlet files go to each of the files shown below and update the values shown in bold. Change myDB2user and myPasswd to the user ID and password that you used to log on to DB2 respectively. Also change the name of the database driver to COM.ibm.db2.jdbc.app.DB2Driver if not already set to this value (this value is case sensitive).

```
<?xml version="1.0"?>
<servlet>
  <page-list>
    <default-page>
      <uri>/IBMWebAS/samples/XtremeTravel/results.jsp</uri>
    </default-page>
    <error-page>
      <uri>/IBMWebAS/samples/XtremeTravel/FakeResults.jsp</uri>
    </error-page>
  </page-list>
  <code>XtremeTravel.FlilghtsServlet</code>
  <description>Flilghts database access servlet for XtremeTravel
sample</description>
  <init-parameter name="URL" value="jdbc:db2:wbsphere"/>
  <init-parameter name="userID" value="myDB2user"/>
  <init-parameter name="driver" value="COM.ibm.db2.jdbc.app.DB2Driver"/>
  <init-parameter name="password" value="myPasswd"/>
</servlet>
```

Figure 405. Changes to <server root>\servlets\XtremeTravel\FlilghtsServlet.servlet

```
<?xml version="1.0"?>
<servlet>
  <page-list>
    <default-page>
      <uri>/IBMWebAS/samples/WebBank/userpage.jsp</uri>
    </default-page>
    <error-page>
      <uri>/IBMWebAS/samples/WebBank/CreateAccountErrorPage.jsp</uri>
    </error-page>
  </page-list>
  <code>WebBank.CreateAccountServlet</code>
  <description>Registered users can create a new account</description>
  <init-parameter name="owner" value="myDB2user"/>
  <init-parameter name="URL" value="jdbc:db2:webbank"/>
  <init-parameter name="userID" value="myDB2user"/>
  <init-parameter name="driver" value="COM.ibm.db2.jdbc.app.DB2Driver"/>
  <init-parameter name="password" value="myPasswd"/>
</servlet>
```

Figure 406. Changes to <server root>\servlets\WebBank\CreateAccountServlet.servlet

```

<?xml version="1.0"?>
<servlet>
  <page-list>
    <default-page>
      <uri>/IBMWebAS/samples/WebBank/RegistrationOutputPage.jsp</uri>
    </default-page>
    <error-page>
      <uri>/IBMWebAS/samples/WebBank/RegistrationErrorPage.jsp</uri>
    </error-page>
  </page-list>
  <code>WebBank.RegistrationServlet</code>
  <description>Allows new users to register with WebBank - their personal
info will be stored in the database</description>
  <init-parameter name="owner" value="myDB2user"/>
  <init-parameter name="URL" value="jdbc:db2:webbank"/>
  <init-parameter name="userID" value="myDB2user"/>
  <init-parameter name="driver" value="COM.ibm.db2.jdbc.app.DB2Driver"/>
  <init-parameter name="password" value="myPasswd"/>
</servlet>

```

Figure 407. Changes to <server root>\servlets\WebBank\RegistrationServlet.servlet

```

<?xml version="1.0"?>
<servlet>
  <page-list>
    <default-page>
      <uri>/IBMWebAS/samples/WebBank/userpage.jsp</uri>
    </default-page>
    <error-page>
      <uri>/IBMWebAS/samples/WebBank/SignonErrorPage.jsp</uri>
    </error-page>
  </page-list>
  <code>WebBank.SignonServlet</code>
  <description>Allows new users to signon to WebBank and access account
info</description>
  <init-parameter name="owner" value="myDB2user"/>
  <init-parameter name="URL" value="jdbc:db2:webbank"/>
  <init-parameter name="userID" value="myDB2user"/>
  <init-parameter name="driver" value="COM.ibm.db2.jdbc.app.DB2Driver"/>
  <init-parameter name="password" value="myPasswd"/>
</servlet>

```

Figure 408. Changes to <server root>\servlets\WebBank\SignonServlet.servlet

```

<?xml version="1.0"?>
<servlet>
  <page-list>
    <default-page>
      <uri>/IBMWebAS/samples/WebBank/userpage.jsp</uri>
    </default-page>
    <error-page>
      <uri>/IBMWebAS/samples/WebBank/TransactionErrorPage.jsp</uri>
    </error-page>
  </page-list>
  <code>WebBank.TransactionServlet</code>
  <description>TransactionServlet for WebBank</description>
  <init-parameter name="owner" value="myDB2user"/>
  <init-parameter name="URL" value="jdbc:db2:webbank"/>
  <init-parameter name="userID" value="myDB2user"/>
  <init-parameter name="driver" value="COM.ibm.db2.jdbc.app.DB2Driver"/>
  <init-parameter name="password" value="myPasswd"/>
</servlet>

```

Figure 409. Changes to <server root>\servlets\WebBank\TransactionServlet.servlet

2. Edit the properties files for the XtremeXML and IBMConnMgrTest samples using the DB2 user ID and password information.

- Copy the file <<ASRoot>>\samples\login.properties to <ASRoot>\servlets. Edit the file and change the database owner and login parameters. Figure 410 shows an extract from the file with the values to change shown in bold. Change DB2owner to the DB2 user ID that you used to create the sample database in step one above. If you want to access DB2 with a user ID other than the user ID and password that WebSphere uses, change the dbUserId and dbPassword values appropriately otherwise leave them as *null*. On Windows NT the WebSphere service does not run under any user ID by default so you must change these values. If you have changed the name of the sample database that you created in step one above to something other than *sample*, then change the dbName parameter to this value.

```

XtremAdvXml.dbOwner=DB2owner
XtremAdvXml.dbUserId=YourDB2UserId
XtremAdvXml.dbPassword=YourDB2Password
...
JDBCServlet.dbOwner=DB2owner
JDBCServlet.dbUserId=YourDB2UserId
JDBCServlet.dbPassword=YourDB2Password
JDBCServlet.dbName=sample

```

Figure 410. Lines to Change in <server root>\servlets\login.properties

- Edit the file <ASRoot>\servlets\IBMConnMgrTestStrings.properties and change the values shown in Figure 411.

```

# UserProfileConfig
upc.db=SAMPLE
upc.owner=DBOwner
upc.userid=DBUser
upc.password=DBPassword

# HTML text
# LOCALIZE THIS
html.title=Test for IBMConnMgrTest Servlet
html.nobodyfound=Nobody named Parker
html.greeting=G'day
# END OF MATERIAL TO LOCALIZE

```

Figure 411. Values to Change in <server home>/servlets/IBMConnMgrTestStrings.properties

Change the upc.db parameter to the name of the sample database (probably sample) and the upc.owner parameter to the owner of the sample database. If you want to log in to DB2 with a user ID and password that is different from the user ID and password that WebSphere runs under, change the upc.userid and upc.password settings appropriately. Otherwise change them to the word null. On Windows NT the WebSphere service does not run under a userid and password by default so you must change these values.

If you want to localize the servlet for your country change the html.* parameters to something in your local language. html.title is used as the title heading on the servlet. html.nobodyfound is the message displayed when no results are returned from the database. html.greeting is the word used to greet someone.

3. Populate the database tables used by the sample applications.

You will need to create data in the database tables used by the sample applications. Perform the following steps:

- Open a DB2 command window.

On Windows NT select **Start -> Programs -> DB2 for Windows NT -> Command Window**.

On AIX open a terminal window and type:

```
. /home/<instance>/sqllib/db2profile
```

where <instance> is the name of the DB2 instance you have created and the space between the . and the rest of the command is significant.

- Change directory to <server root>\samples\XtremeTravel\database
- Run Createdb.bat (see Figure 412 on page 454). If all goes well you should see the following message near the bottom of the output:


```
SQL3149N "12" rows were processed from the input file. "12" rows were
successfully inserted into the table. "0" rows were rejected.
```

```
Number of rows read          = 12
Number of rows skipped       = 0
Number of rows inserted      = 12
Number of rows updated       = 0
Number of rows rejected      = 0
Number of rows committed    = 12
```

- Change directory to <server root>\samples\WebBank\database
- Run createdb.bat (See Figure 413 on page 454). On Windows NT if all goes well you should see the following message near the bottom of the output:

```
D:\WebSphere\AppServer\samples\WebBank\database>db2 "create table customer (user
name varchar(12) not null primary key, pass varchar(12) not null, firstname varc
har(40), lastname varchar(40), street varchar(100), city varchar(50), state varc
har(2), zip varchar(5), phone varchar(15), email varchar(50))"
DB20000I The SQL command completed successfully.
```

```
D:\WebSphere\AppServer\samples\WebBank\database>db2 "create table bankaccount (u
sername varchar(12) not null, pass varchar(12), accounttype varchar(32) not null
, balance Decimal (15,2), primary key (username, accounttype))"
DB20000I The SQL command completed successfully.
```

On AIX the same messages display this way:

```
DB20000I The SQL command completed successfully.
DB20000I The SQL command completed successfully.
```

You will see some error messages if this is the first time you have run these programs as they try to delete databases and tables that are not there, in preparation for creating the new ones. If you see the completion messages shown above, then all has completed successfully.

```

db2start
db2 "create database wbsphere"
db2 "force application all"

db2 "connect to wbsphere"
db2 "drop table flights"
db2 "create table flights (id CHAR(4) NOT NULL PRIMARY KEY, DepartFrom_1
CHAR(25), DepartTime_1 CHAR(25), ArriveIn_1 CHAR(25), ArriveTime_1
CHAR(25), Airline_1 CHAR(25), DepartFrom_2 CHAR(25), DepartTime_2 CHAR(25),
ArriveIn_2 CHAR(25), ArriveTime_2 CHAR(25), Airline_2 CHAR(25))"
db2 "import from flights.txt of del insert into flights (id, DepartFrom_1,
DepartTime_1, ArriveIn_1, ArriveTime_1, Airline_1, DepartFrom_2,
DepartTime_2, ArriveIn_2, ArriveTime_2, Airline_2)"
db2 "connect reset"

```

Figure 412. Xtreme Travel Createdb.bat

```

db2 drop database webbank
db2 create database webbank
db2 force application all
db2stop
db2start

db2 connect to webbank

db2 drop table customer
db2 drop table bankaccount

db2 "create table customer (username varchar(12) not null primary key, pass
varchar(12) not null, firstname varchar(40), lastname varchar(40), street
varchar(100), city varchar(50), state varchar(2), zip varchar(5), phone
varchar(15), email varchar(50))"
db2 "create table bankaccount (username varchar(12) not null, pass
varchar(12), accounttype varchar(32) not null, balance Decimal (15,2),
primary key (username, accounttype))"

```

Figure 413. Web Bank Createdb.bat

4. Create an alias schema for the flights table for the XtremeAdventures sample.

If you have created these databases from a user ID other than db2admin the XtremeAdventures sample will not be able to find the flights table. This is because it will be looking under the db2admin schema and your table was created under a schema matching your user ID. To fix this you will need to create an alias schema for db2admin.

On Windows NT use the DB2 Control Center to select the alias folder for the WEBSPHERE database. Right click and select **Add** to enter the information shown in Figure 414 on page 455.

On AIX you can use the Web control center in the same manner or type the following commands at a command prompt after running the db2profile script:

```

db2 connect to wbsphere
db2 create alias db2admin.flights for <userid>.flights

```

where <userid> is the user ID you used to run the <server root>\samples\WebBank\database\createdb.bat script in step 3 on page 452.

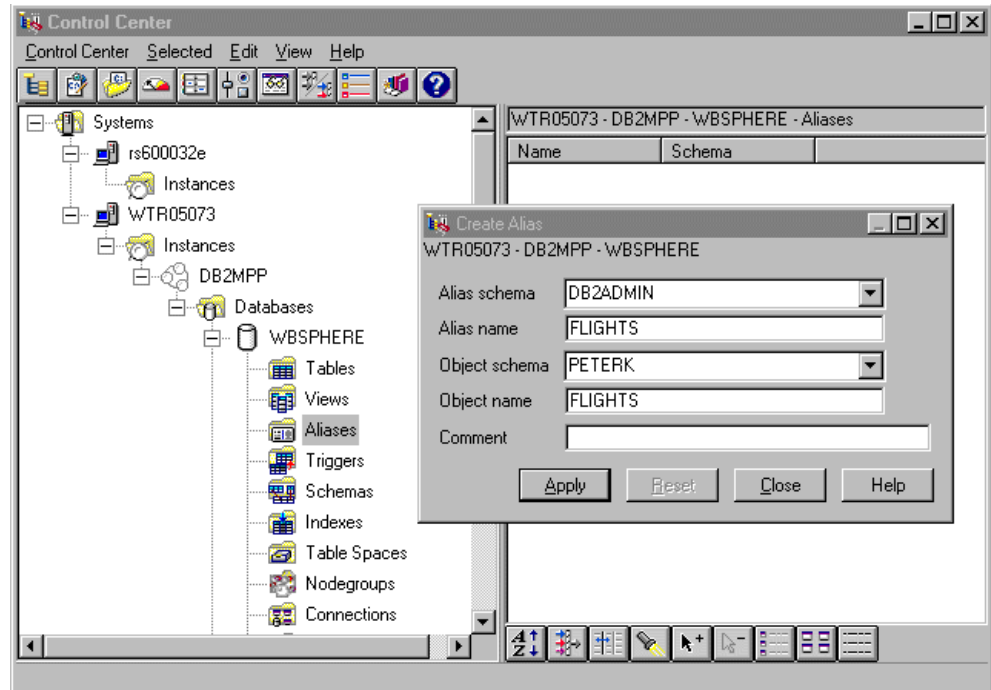


Figure 414. Creating a Schema Alias

A.1 Other Configuration Steps

The following are some miscellaneous configuration steps that you will need to perform in order to get the samples running:

- To enable the form processing servlet to find its seminar details you must copy <server root>/samples/FormProcessingServlet/seminar.txt to the root directory of your web server's document hierarchy.
- To get the XML version of the Xtreme Adventures Web site working you will need to add the xml4j.jar file to the Application Server Classpath. See 4.2.1, "Setting Up the Environment" on page 188 for details on how to perform this task.

After completing the configuration steps for the samples that you want to run, you should now be able to run the various samples. We had some trouble getting a number of the samples to work. All of the errors we managed to track down have been included in the setup instructions, but there were a number of strange errors that were not reproducible. If you are experiencing problems, try shutting everything down and rebooting. Usually this will fix the problem. Also try executing the samples from a different machine or a different browser. If you run into a problem that is reproducible, try one of the debugging techniques from Chapter 8, "Problem Determination" on page 411.

The samples can be divided into three groups:

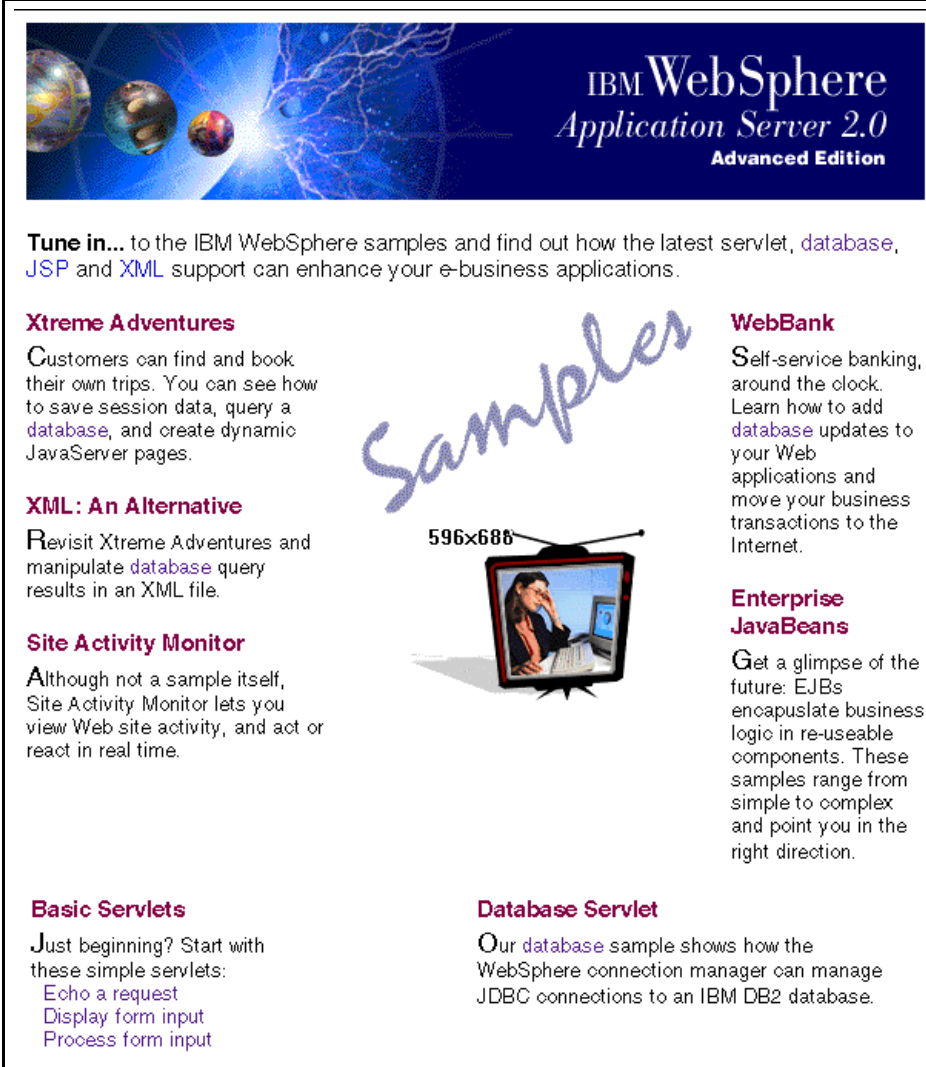
1. Samples from the samples Web page
2. EJS samples

3. Other samples

Read one of the following sections for tips on the sample you want to run:

A.1.1 Samples From the Samples Web Page

Go to the samples Web page at <http://<your server>/IBMWebAS/samples/> (see Figure 415 on page 456).



**IBM WebSphere
Application Server 2.0
Advanced Edition**

Tune in... to the IBM WebSphere samples and find out how the latest servlet, [database](#), [JSP](#) and [XML](#) support can enhance your e-business applications.

Xtreme Adventures
Customers can find and book their own trips. You can see how to save session data, query a [database](#), and create dynamic JavaServer pages.

XML: An Alternative
Revisit Xtreme Adventures and manipulate [database](#) query results in an XML file.

Site Activity Monitor
Although not a sample itself, Site Activity Monitor lets you view Web site activity, and act or react in real time.

Basic Servlets
Just beginning? Start with these simple servlets:
[Echo a request](#)
[Display form input](#)
[Process form input](#)

Database Servlet
Our [database](#) sample shows how the WebSphere connection manager can manage JDBC connections to an IBM DB2 database.

WebBank
Self-service banking, around the clock. Learn how to add [database](#) updates to your Web applications and move your business transactions to the Internet.

Enterprise JavaBeans
Get a glimpse of the future: EJBs encapsulate business logic in re-useable components. These samples range from simple to complex and point you in the right direction.

Figure 415. The WebSphere Samples Page

Click the links for the following samples and follow the instructions.

A.1.1.1 Xtreme Adventures

The Xtreme Adventures sample models a simple travel agent Web site. On each of these pages you can click the **explanation** or **code** buttons to see more information about what's on the screen and how it works.

- Click the **register now!** link.
- Enter details on the form and click **Submit**. It doesn't matter what you type, there is no field validation of these details.

- Click either the rock climbing or skiing pictures.
- Pick a holiday package and click the link.
- Go down to the bottom of the page and click **Flight Schedule**.
- Pick either Atlanta or Chicago from the drop down list box and click the **Find Flights** button.
- Scroll down to the bottom of the page and click **Book It**.
- Experiment and click the **explanation** and **code** buttons to see how the site works.

A.1.1.2 Xtreme XML

Xtreme XML is a variation on the Xtreme Travel Web site but it uses XML under the covers to do the work. There are actually three samples contained here that perform the flight lookup portion of the XtremeTravel Web site using XML to format the result. Each sample uses a different method to process the XML. A tutorial on methods of processing XML is beyond the scope of this book but more reference information can be found at <http://www.software.ibm.com/xml>.

The sample can be accessed in the following way:

- From the samples page click **XML: An Alternative**.
- Click either **XML for Java parser**, **SAXDriver parser** or **ElementHandler parser**.
- Enter a name in the **Your name:** field.
- Select a city from the drop down menu.
- Click the **Submit** button under the field to see the flight information.
- More information on what is going on can be found by clicking the **Developing XML-based applications** link on the first page which links to <http://<your web server>/IBMWebAS/doc/howto/itxml4j.html>.

A.1.1.3 WebBank

WebBank gives you a chance to feel like an instant virtual millionaire right at your own computer! It implements a simple online banking service that allows accounts to be added and deposits and withdrawals to be made. The account information is maintained in the WebBank database created previously. As with the Xtreme Adventures sample, the WebBank sample includes an extensive commentary on how it is constructed and what is going on. Simply click the **explanation** or **code** links to access the information.

The following are some suggestions for using the WebBank sample:

- Click the **WebBank** link on the samples page. The WebBank initial page comes up in another window.
- Click the **register** link to register a new customer.
- Fill in all of the fields and enter a user ID and password and click the **Submit** button.
- The Registration Successful page appears. Click the **here** link near the bottom to go to the main banking page.
- Select an account type to create from the drop-down menu (Cheque, Savings or Money Market). Click **Submit** to create the account.

- Deposit a million dollars into your new account by entering data into the fields in the Make a Transaction section.
- Create another account and deposit some money.
- Withdraw money from your accounts.
- Have a look at the documentation for the sample by clicking the **explanation** and **code** links.

A.1.1.4 IBM Connection Manager Test Servlet

This servlet, while not terribly exciting visually, demonstrates the IBM Connection Manager performing connection pooling. To run the servlet:

- Click the **Database Servlet** from the samples page. Another browser window will come up.
- Click **Run IBMConnMgrTest**. Yet another browser window should come up with a greeting for JOHN Parker. The form of the greeting derives from the text you entered in the database setup for the `html.greeting` parameter.
- If something goes wrong you should get the error message that you entered for the `html.nobodyfound` parameter in the database setup.

A.1.1.5 Request Info Servlet

The Request Info servlet can be accessed by clicking the **Echo a Request** link near the bottom of the samples page and then clicking the **Run the RequestInfoServlet** link on the other browser window that appears. The servlet will display some information about the http request that was passed to it. Documentation is available by clicking **Review the logic** and **Look at the source code** links on the RequestInfoServlet browser window.

A.1.1.6 Form Display Servlet

The Form Display servlet can be accessed by clicking the **Display form input** link near the bottom of the samples page and then clicking the **Run the FormDisplayServlet** link on the other browser window that appears. The servlet will display a form which allows you to enter some seminar booking details in the fields and when you press **Submit**, will echo the parameter names and values back. Documentation is available by clicking **Review the logic** and **Look at the source code** links on the FormDisplayServlet browser windows.

A.1.1.7 Form Processing Servlet

The Form Processing servlet can be accessed by clicking the **Process form input** link near the bottom of the samples page and then clicking the **Run the FormProcessingServlet** link on the other browser window that appears. The servlet will display a form which allows you to enter some seminar booking details in the fields. When you press **Submit** it will display the results of your attempt to book the seminar based on your preferences. Documentation is available by clicking **Review the logic** and **Look at the source code** links on the FormProcessingServlet browser windows.

Appendix B. Special Notices

This publication is intended to help webmasters set up the WebSphere infrastructure for their Web applications. The information in this publication is not intended as the specification of any programming interfaces that are provided by the WebSphere Application Server. See the PUBLICATIONS section of the IBM Programming Announcement for WebSphere Application Server Standard Edition and WebSphere Application Server Advanced Edition for more information about what publications are considered to be product documentation.

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 program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

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 the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 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, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

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

AIX	AS/400
AT	CICS
CT	DB2
eNetwork	Home Director
IBM	JCentral
MQ	MQSeries
Netfinity	OS/2
OS/390	OS/400
RS/6000	S/390
SP	System/390
TXSeries	VisualAge
VM/ESA	WebSphere
XT	400

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

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

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

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

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

UNIX is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.

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

Appendix C. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

C.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How to Get ITSO Redbooks” on page 465.

- *VisualAge for Java Enterprise Version 2: Data Access Beans - Servlets - CICS Connector*, SG24-5265
- *Using VisualAge for Java Enterprise Edition Version 2 to Develop CORBA EJB Applications*, SG24-5276
- *Programming with VisualAge for Java Version 2*, SG24-5264
- *IBM WebSphere Performance Pack Usage and Administration*, SG24-5233
- *Connecting the Enterprise to the Internet with MQSeries and Visual Age for Java*, SG24-2144

C.2 Redbooks on CD-ROMs

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at <http://www.redbooks.ibm.com/> for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	Collection Kit Number
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbooks Collection	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849
Netfinity Hardware and Software Redbooks Collection	SK2T-8046
RS/6000 Redbooks Collection (BkMgr Format)	SK2T-8040
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037

C.3 Other Publications

These publications are also relevant as further information sources:

- *Quick Beginnings for DB2 Extended Enterprise Edition*, S99H-8314
- *MQSeries Application Programming Guide*, SC33-0807
- *MQSeries using Java*, SC34-5456

C.4 Web Sites Referenced in This Book

<http://www.ibm.com/java/jdk/>
<http://www.ibm.com/developer/xml/>
<http://www.ibm.com/xml>
<http://www.redbooks.ibm.com>

<http://www.developer.ibm.com/java>
<http://www.alphaworks.ibm.com>

<http://www.software.ibm.com>
<http://www.software.ibm.com/ad/vajava>
<http://www.software.ibm.com/ts/mqseries>
<http://www.software.ibm.com/webservers>
<http://www.software.ibm.com/webservers/httpservers/download.html>
<http://www.software.ibm.com/webservers/analysis>
<http://www.software.ibm.com/webservers/studio/doc/wsguide.pdf>
<http://www.software.ibm.com/ebusiness/library.html>
http://www.software.ibm.com/ebusiness/arch_overview.html
<http://www.software.ibm.com/ebusiness/appservsw>
<http://www.software.ibm.com/ebusiness/pm.html#Servlets>
<http://www.software.ibm.com/developer/library/tutorial-prog/writing.html#dtds>
<http://www.software.ibm.com/ad/cb>
[http://www.software.ibm.com/ebusiness/pm.html#Java Server Pages](http://www.software.ibm.com/ebusiness/pm.html#Java_Server_Pages)
http://www.software.ibm.com/xmlhttp://www.software.ibm.com/ebusiness/connector_s.html
<http://www.software.ibm.com/ebusiness/connectors.html>

<http://www7.software.ibm.com>
<http://www7.software.ibm.com/vad.nsf/Data/Document3172?OpenDocument&SubMast=1>

<http://java.sun.com>
<http://java.sun.com/products/ejb/>
<http://java.sun.com/products/ejb/developers-guide.pdf>
<http://java.sun.com/products/jsp/>
<http://java.sun.com/products/jdbc>
<http://java.sun.com/products/servlets/index.html>
http://java.sun.com/products/servlet/2.1/api/javax.servlet.http.HttpSession.html#_top_
<http://java.sun.com/products/jndi/tutorial/index.html>
<http://java.sun.com/products/jdk/rmi/index.htm>
<http://java.sun.com/products/jdk/1.1/docs/tooldocs/win32/index.html>
<http://developer.java.sun.com>

<http://www.javasoft.com>
<http://www.javasoft.com/beans/docs>
<http://www.javasoft.com/products/jsp/index.htm>
http://www.javasoft.com/products/jsp/JSP-1_0-public-draft-1.pdf

<http://www.netscape.com/servers>
http://www.netscape.com/newsref/std/cookie_spec.html

<http://www.lotus.com>
<http://www.notes.net/notesua.nsf>

<http://www.w3c.org>
<http://www.w3.org>
<http://www.w3.org/TR/WD-xs>
<http://www.w3.org/TR/WD-logfile>
<http://www.w3.org/TR/WD-xsl/>
<http://www.w3.org/TR/PR-DOM-Level-1/>
<http://www.w3.org/TR/PR-DOM-Level-1/java-language-binding.html>
http://www.w3.org/TR/REC-xml_names/
<http://www.w3.org/XSL/Transform/1.0>
<http://www.w3.org/TR/1999/WD-xslt-19990421.html>

<http://www.xml.com>

<http://www.xml.com/xml/pub/Style>
<http://www.xml.com/xml/pub/1999/01/namespaces.html>
<http://www.xml.com/xml/pub/1999/01/walsh3.html>
<http://www.xml.com/axml/testaxml.htm>

<http://info.webcrawler.com>
<http://www.webtechniques.com>
<http://www.sqlj.org>
<http://www.oracle.com>
<http://www.whatis.com/iiop.htm>
<http://www.omg.org/corba/beginners.html>
<http://www.microstar.com>
<http://www.microstar.com/sax.html>
<http://www.megginson.com/SAX/>
<http://www.otp.org>
<http://www.apache.org>

<http://www.netobjects.com>
<http://www.netobjects.com/products/html/nbb1.html>
<http://www.netobjects.com/products/html/nsb3.html>
<http://www.netobjects.com/products/html/nf3i.html>

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** <http://www.redbooks.ibm.com/>

Search for, view, download or order hardcopy/CD-ROM redbooks from the redbooks web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this redbooks site.

Redpieces are redbooks in progress; not all redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

Send orders via e-mail including information from the redbooks fax order form to:

	e-mail address
In United States	usib6fpl@ibmmail.com
Outside North America	Contact information is in the "How to Order" section at this site: http://www.elink.ibm.ibm.com/pbl/pbl/

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the "How to Order" section at this site: http://www.elink.ibm.ibm.com/pbl/pbl/

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the "How to Order" section at this site: http://www.elink.ibm.ibm.com/pbl/pbl/

This information was current at the time of publication, but is continually subject to change. The latest information for customer may be found at <http://www.redbooks.ibm.com/> and for IBM employees at <http://w3.itso.ibm.com/>.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and redbooks by accessing the IBM Intranet Web site at <http://w3.itso.ibm.com/> and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may also view redbook, residency, and workshop announcements at <http://inews.ibm.com/>.

Index

A

Administration interface 3
analyzing links 1
Apache Web server 2, 34
apachectl 71

B

bulletins 286

C

CAE 52, 53
callPage 140
chaining 14, 17, 108
CLASSPATH 31, 32, 54, 70, 89, 365, 381
classpath 106, 278, 351, 353, 359, 427, 434
clustering 248, 251
connection handle 317
Connection Manager 263, 265
connection manager 306, 307, 310, 311, 315, 318, 323
connection URL 350, 351, 357
container 192, 195
Content Analysis 395, 396
Cookies 221, 239
cookies 247
CORBA 3, 7, 86, 197, 207, 209, 210, 212, 356, 358

D

Data Type Definition 166
databases 5
DB2 UDB V5.2 2
debug.properties file 419, 420, 422, 427, 428, 434
declaration 341
deployment 2, 187, 190, 197, 200, 204, 207, 216
directive 34, 140
Directory paths 1
document root 140
Document Structure Services 153, 154, 158, 165
domino.cnf 62, 78
DriverManager 335
dynamic content 13
dynamic HTML 5, 18
dynamic XML 153

E

EJB 2, 6, 19, 86, 199, 331, 356, 361
EJB Container 19
EJB container 189, 190
EJB containers 187
EJB deployment 209
EJS 90, 187, 412, 445
EJS process monitor 213
element handler 159
element handlers 158
Enterprise Edition 2

Enterprise Java Services 3
Entity Beans 20
entity beans 216
event driven parsing 155
explicit indexing 139

F

filtering 14, 17, 110, 427, 433
firewall 219, 378

G

go46.dll 41, 62

H

helper class 210, 211, 212
HTML document root 94
HTTP document root 125, 235, 259, 269, 284, 286, 291, 300
HTTP protocol 2
HTTP requests 2
httpd.cnf 38, 40, 62, 64, 78, 80
httpd.conf 36, 64, 71, 80
httpd.properties 110
HttpRequest 15
HttpResponse 15
HttpServletRequest 129, 141
HttpServletResponse 129

I

IBM HTTP Server 2, 32, 70
IBM jCentral 30
IBM VisualAge for Java 4
ibm_app_server_module 71
IIOF 7
IIS 47
iis20.dll 51
introspected 131
invoker servlet 103

J

JAR 6, 96, 103, 125, 154, 199
Java Foundation Classes 198
JDBC 8, 32, 52, 84, 175, 314, 326, 332, 337, 390, 442
JDBC result set 336
JDBCAccess 354
JDK 29, 30, 32, 64, 67, 68, 387, 436
Jet 197, 198, 207
JNDI 3, 7, 208, 213, 359
JSP 2, 18, 86, 91, 120, 140, 356, 440
JSP directives 127, 128

L

Linux 349
listener definition 381

listener service 381
Loaded Servlets Monitor 126
log files 394, 400, 401, 421
logs 411
LSD 187, 412

M

management 2, 187
MIME type 17
MIME-Type 112
MIME-type 113
MMC 48
MQSeries 362, 364

N

ns35.dll 44

O

obj.conf 64, 80

P

PATH 54
path 353, 359, 381
Persistence 8
persistence 20, 231, 331, 358
plug-in 34, 40, 43, 51, 62, 64, 70, 71, 73, 78
PNS 187, 412
Project Center 392, 396
Project tree 392, 409
property file 360
property values 101

R

Redeployment Selection 194
reloadable servlet directory 106
remote loading 124
RMI 7
Robot Exclusion Protocol 395

S

SAX parser 158
SAX parsing 156
SER 96
servlet 91, 374, 414
Servlet Service 419
Servlets 5, 13
servlets 145, 369
session bean 358
Session Beans 20
session tracking 225
SHTML 92
Site Analyzer 29
site policies 394
SQL 272, 315, 335
SQLJ 32, 52, 89, 332, 338, 340, 349
SSL 2, 376

standard error log 412
standard output log 413
stateful 20
stateless 20, 221
static 153
static HTML 12
style sheet 153, 165, 169, 179
System.err 417, 419
System.out 417, 419

T

three-tier 5
token 155
trace log 414
tracer 420, 428, 431
tracers 422, 424, 425, 430, 445
transaction systems 5
Type 2 driver 337, 346, 350
Type 3 driver 350

U

URL rewriting 239, 246, 247
Usage Analysis 400, 401, 404, 408
utility class 210

V

VA Java 29
Virtual servers 45
VisualAge for Java 3, 433, 445

W

Web robot 394, 395, 396
WebSphere Performance Pack 1
WebSphere Servlet Service 192, 195, 196
WebSphere Site Analysis 1
WebSphere Studio 1, 4, 86

X

XML 2, 6, 91, 152, 207
xml4j 154
XSL 164, 177
XSL stylesheets 6

ITSO Redbook Evaluation

WebSphere Application Servers: Standard and Advanced Editions
SG24-5460-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at <http://www.redbooks.ibm.com>
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?

Customer **Business Partner** **Solution Developer** **IBM employee**
 None of the above

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes___ No___

If no, please explain:

What other redbooks would you like to see published?

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

SG24-5460-00

Printed in the U.S.A.

