WebSphere Application Server for OS/390

WebSphere Application Server Standard Edition Planning, Installing, and Using

Version 3.5

GC34-4835-05

WebSphere Application Server for OS/390

WebSphere Application Server Standard Edition Planning, Installing, and Using

Version 3.5

GC34-4835-05

Note

Before using this information and the product it supports, be sure to read the general information under “Notices” on page K-1.

Sixth Edition (July, 2003)

This edition applies to WebSphere Application Server for OS/390 Version 3.5, product number 5655-A98, and to all subsequent releases and modifications until otherwise indicated in new editions.

The most current versions of the WebSphere Application Server for OS/390 publications are located at the following web site:

`http://www.ibm.com/software/websphere/appserv/zos_os390/library.html`

You can E-mail your comments to waseedoc@us.ibm.com or fax them to 919–254–0206.

Contents

Summary of Changes

This book is available in softcopy formats only. The most current version is available in PDF format on the following Web sites:

```
http://www.ibm.com/software/websphere/appserv/zos_os390/library.html
```

and

```
http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi
```

Updates, June 2003

The following changes have been made to this publication since the last revision:

- The fully qualified name of the default was.conf file, *applicationserver_root*/**AppServer/properties/was.conf** specified in section "Configuring the Application Server" on page 2-5 of Chapter 2 has been corrected.
- The following changes were made to Chapter 3:
  - The section, "Improving JSP compile time" on page 3-25, was added.
  - An additional note describing when to use the appserver.extraparm=-Dcom.sun.jsp.useBeanConstructorMethod=true property was added to section "Including Web components in a Web application" on page 3-5.
  - A description of when to set the new webapp.<webapp-name>.servlet.jsp11.initargs=allowwebinf was.conf file property to true was added to section "Passing init-parameters to a servlet" on page 3-26.
  - The section, "Pre-compiling JSPs" on page 3-22, was updated with the new optional parameters that APARs PQ68506 and PQ67191 added to the jsp10BatchCompiler.sh and jsp11BatchCompiler.sh shell scripts.
- The following changes were made to Chapter 4:
  - A description of the new jdbcconnpool.<pool-name>.provider property, provided in APAR PQ66335 has been added to the section "Setting up JDBC connection pools" on page 4-4.
  - A note was added to section "How servlets use the JDBC 2.0 Standard Extension API" on page 4-2 describing what to do if a naming conflicts occurs while using the com.ibm.ejs.ns.jndi.CNInitialContextFactory class.
  - The example in section "Migrating Connection Manager Code to use the JDBC Standard 2.0 Extension APIs" on page 4-8 was corrected.
- Section "In-memory session pools" on page 5-7 in Chapter 5 was updated with the correct name of the method used to indicate that a returned session is invalid.
- A description of the new jdbcconnpool.<pool-name>.provider property, that was added to the was.conf file in APAR PQ66335 has been added to Appendix B, "was.conf file template", on page B-1.
- Technical inaccuracies in section "Enabling the Application Server to locate, and communicate with, DB2" on page E-3 in Appendix E have been corrected.

Updates, September 2002

The following changes have been made to this publication since the last revision:

- Section "Configuring a Web server to host an Application Server" on page 2-1 in Chapter 2 was updated to indicate that the ServiceSync On directive should be added to the HTTP Server's configuration file if a servlet, by any means, sets an HTTP response code, and the client does not receive the expected response code.
- Updates have been made to the following sections of Chapter 3:
  - A new section, "Compiling JSP level 1.0 or level 1.1 source files" on page 3-22, documents new function that was added in response to APAR PQ61925.
  - A description of how to leverage the newer 32K branch mechanism for JSPs that throw BranchTooLarge exceptions, that is available with SDK V1.3, was added to the topic "Java Server Pages (JSPs) and JHTML files" in section "Including Web components in a Web application" on page 3-5.
  - A migration consideration has been added to the section"Passing init-parameters to a servlet" on page 3-26.
  - The table in section "Compiling servlets" on page 3-26 has been updated.
- Technical inaccuracies in section "Enabling the Application Server to locate, and communicate with, DB2" on page E-3 in Appendix E have been corrected.
- A new Appendix Appendix G, "Messages EJS3002I - EJS3087E", on page G-1 has been added.

Updates, July 2002

The following changes have been made to this publication since the last revision:

- The following changes were made to Chapter 2:
  - Information was added to the description of the ServerInit directive contained in section "Configuring a Web server to host an Application Server" on page 2-1.
  - A new section, "Configuring the Application Server to use the Xerces.jar and Xalan.jar files distributed with Apache" on page 2-10, has been added.
  - The description of the request mapping behavior differences contained in the table in "Maintaining compatibility with existing applications" on page 2-9 has been clarified.
- Updates have been made to the following sections of Chapter 3:
  - "Configuring servlet chaining" on page 3-29
  - "Defining and deploying Web applications" on page 3-3
  - "Including Web components in a Web application" on page 3-5
  - "Using Web applications contained in War files" on page 3-11
- Corrections were made to the classpath description in section "Deploying a Web application to the Application Server" on page 3-7.
- A description of the new was.conf configuration property, appserver.permissions, was added to Appendix B, "was.conf file template", on page B-1.
- Corrections were made to the description of the content of the Web server's httpd.envvars configuration file in the section "Enabling the Application Server to locate, and communicate with, DB2" on page E-3.

Summary of Changes
for GC34-4835-02
as Updated April 2002

- Information was added to "Deploying a Web application to the Application Server" on page 3-7 indicating that JDK/SDK .jar files containing classes required by your servlets, such as the recordio.jar file which contains the JRIO classes, must be included in the appserver.classpath rather than in the deployedwebapp.classpath.

- The information on servlet chaining was clarified and moved from Appendix E to Chapter 3.
- Changes have been made to "Enabling the Application Server to locate, and communicate with, DB2" on page E-3.

Summary of Changes
for GC34-4835-01
as Updated October 2001

- The following changes have been made to **Chapter 1. Planning for Installation**:
  - New information has been added to the section "Deploying Components generated by WebSphere Studio" on page 1-10.
  - DB2 UDB for OS/390 and z/OS Version 7 has been added as a version of DB2 that can be used with V3.5 of the Application Server.
  - A new section, "IBM Distributed Debugger and Object Level Trace support" on page 1-6, has been added
  - A description of the CICS Connector support, that is now available via Version 4.0 of the CICS Transaction Gateway product, has been added.
  - Information was added to sections "Compatible OS/390 releases" on page 1-1, and "Required OS/390 Web server" on page 1-1, indicating that Version 3.5 can be used with z/OS.
  - The section "XML Document Structure Services" on page 1-3 has been updated to reflect that XML parser level 2.0 is also supported.
  - Instructions not to replace the copy of the databeans.jar file shipped with the Application Server with the copy that is shipped with the Studio product have been added to the section"Deploying Components generated by WebSphere Studio" on page 1-10.
- A new step has been added to the section "Configuring a Web server to host an Application Server" on page 2-1 in **Chapter 2. Installing and Customizing the Application Server**.
- The following changes have been made to **Chapter 3. Defining virtual hosts and Web applications**:
  - Additional information has been added to the section "Using Web applications contained in War files" on page 3-11.
  - The section "Setting the Application Server (JVM) and Web application classpaths" on page 3-14 has been updated to indicate which classpath ASCII and EBCDIC files need to be placed in.
  - The section "Compiling servlets" on page 3-26 to indicate that the system CLASSPATH environment variable must includes your SDK rt.jar file.
  - A description of how reloading affects the collection of session data has been added to the section "Class loading and optional reloading" on page 3-21.
  - The table of required JAR files in the section "Compiling servlets" on page 3-26 has been corrected.
  - A new section, "Placing .property files in appropriate directories" on page 3-16, was added.
  - Incorrect references to a webapp.xml file, contained in Chapter 3, "Defining virtual hosts and Web applications", on page 3-1, have been changed to <webapp-name>.webapp file.
  - The example in section ″Configuring a virtual host″ has been corrected.
  - The description of the deployedwebapp.<*webapp-name*>.classpath=<*value*> property has been updated to indicate that it is required for all Web applications being deployed.

- Additional classpath information was added to “Deploying a Web application to the Application Server” on page 3-7.

- The following changes have been made to **Chapter 4. Accessing relational databases**:
  - Information has been added to the section “Setting up JDBC connection pools” on page 4-4.
  - The java.util.Hastable import statement was added to the coding sample in the section “How servlets use the JDBC 2.0 Standard Extension API” on page 4-2.
- The following changes have been made to **Chapter 5. Session tracking**:
  - In the section “Session security” on page 5-1, references to the getUserName method have been changed to getRemoteUser method.
  - Section “Configuring a session cluster” on page 5-4 has been updated to include a description of the index that needs to be included in the DB2 table that will be used to collect session data.
- The following changes have been made to **Appendix B. was.conf file template**:
  - The descriptions of the **jdbcconnpool.**<*pool-name*>**.connectionidentity**, **appserver.compliance.mode**, and **appserver.nodetach=true|false** properties have been updated.
  - A description of the **inline.comment** property was added.
  - The **webapp.**<*webapp-name*>.**servletmapping=**<*URI-pattern*> property has been changed to make it compliant with the J2EE Specification.
- **Appendix C. default_global.properties file** has been updated to reflect changes to this file that were included in a recent PTF.
- Two new appendices have been added:
  - **Appendix E. Passing init-parameters to a servlet and configuring servlet chaining**
  - **Appendix G. Using the Connection Manager APIs**. This appendix has been added for users who are migrating from a previous version of the Application Server. Because the Connection Manager may not be supported in future Application Server releases, these APIs should **not** be used in newly developed servlets; the Java Database Connectivity (JDBC) API described in Chapter 4, “Accessing relational databases”, on page 4-1 should be used to establish connectivity to a relational database.

Welcome!

WebSphere Application Server Version 3.5 is a new release of the WebSphere Application Server Standard Edition feature of the WebSphere Application Server for OS/390 product. This new release includes:

- Java 2 Software Development Kit (SDK) 1.3 support
- Support for Servlets written to the Java Servlet Version 2.2 Specification
- Support for JavaServer Pages written to the JavaServer Pages Version 1.1 Specification level
- Utilities for importing a Web application that is packaged as a Web Application Archive (.war) file into the Application Server for execution.

See Chapter 2, "Installing and customizing the Application Server", on page 2-1 for information on how to use these new capabilities.

The process for configuring and operating the Application Server is unchanged from Version 3.02 of the Standard Edition product. If you are migrating from Version 3.02, you should be able to configure the Version 3.5 Application Server by making a few small changes to your existing Version 3.02 was.conf files.

In addition to maintaining consistent configuration support, the Application Server provides a servlet compatibility mode. In compatibility mode, the application server will not enforce portions of the Java Servlet Specification Version 2.2 that are not compatible with the Version 2.1 Specification. Therefore, you are not required to make changes to existing servlets that were deployed in your Version 3.02 Application Server to accommodate changes imposed by the new specification level.

The Application Server continues to provide support for JavaServer Pages written to the 0.91 and 1.0 specification levels. The desired specification level can be specified within each Web application definition. Web applications that were deployed in version 3.02 of the Application Server that contain JavaServer Pages written to the 0.91 and 1.0 level do not have to be changed for this version of the Application Server.

Version 3.5 of the WebSphere Application Server Standard Edition feature requires the use of the Java Development Kit Version 1.3, which is shipped with the product.

This new release of the Standard Edition feature no longer includes IBM WebSphere Application Server Site Analyzer. This element is now available as a separately orderable product. See the following URL for information on how to order this product.

`http://www.ibm.com/software/webservers/siteanalyzer/`

This book only includes information on the WebSphere Application Server Standard Edition Version 3.5 element of the Standard Edition feature. For the most current Java documentation, go to URL:

`http://www.ibm.com/servers/eserver/zseries/software/java/`

The most current version of this book and related Version 3.5 product documentation for the z/OS platform is available on the WebSphere Application Server Web site library page at URL:

```
http://www.ibm.com/software/websphere/appserv/zos_os390/library.html
```

Your feedback is important in helping to provide the most accurate and highest quality information. If you have any comments about this book or any other WebSphere Application Server publication, E-mail them to waseedoc@us.ibm.com. Be sure to include the name of the book, the document number of the book, the version of WebSphere Application Server for z/OS, and if applicable, the specific location of the information on which you are commenting (for example, a page number or table number).

Application Server product information

WebSphere Application Server Standard Edition Version 3.5, hereafter referred to simply as the Application Server, is a plug-in DLL that runs within the IBM HTTP Server's address space. The IBM HTTP Server is referred to hereafter as the Web Server. The Application Server provides an environment for defining and deploying Java Web applications.

This version of the Application Server can only be obtained as a ServerPac or CBPDO:

- A ServerPac is an entitled software delivery package consisting of products and service for which IBM has performed the SMP/E installation steps and some of the post-SMP/E installation steps. To install the package on your system and complete the installation of the software that it includes, you use the CustomPac Installation Dialog. *ServerPac: Using the Installation Dialog*, SC28-1244, describes how to do this. The ServerPac installation performs a full feature replacement and installs the complete WebSphere Application Server Standard Edition V3.5 feature. Use an order checklist (available from your IBM representative, the OS/390 Web site, or the configurator function of IBMLink) to order the ServerPac for the Version 3.5 feature.
- A CBPDO (Custom-Built Product Delivery Option) is an entitled software delivery package consisting of un-installed products and un-integrated service. You must use SMP/E to install the WebSphere Application Server Standard Edition for OS/390 feature and its service. Use an order checklist (available from your IBM representative, the OS/390 Web site, or the configurator function of IBMLink) to order the CBPDO for the Version 3.5 feature.

For the latest Application Server product offerings, information, and news, go to the WebSphere Application Server for OS/390 Web site at URL:

```
  http://www.ibm.com/software/websphere/appserv/os390.html
```

For WebSphere Application Server Standard Edition installation information, see the *WebSphere Application Server for OS/390 Version 3.5 Program Directory* that is shipped with the product. This document is also available in PDF format on the WebSphere Application Server for OS/390 Web site library page.

Additional product information is contained in the *WebSphere Troubleshooter for OS/390*.

The *WebSphere Troubleshooter for OS/390* is available on the Web and provides the most current debugging and tuning tips for the Application Server. To access the *Troubleshooter*, go to URL:

```
http://www.ibm.com/software/websphere/httpservers/troubleshooter.html
```

Documentation formats

The *WebSphere Application Server Standard Edition Version 3.5 for OS/390 Program Directory* is available in both hardcopy and PDF format.

The *WebSphere Troubleshooter for OS/390* is available in HTML format only.

The *WebSphere Application Server Standard Edition Planning, Installing and Using*, GC34-4835, is available in HTML and PDF formats.

To access the most current Application Server documentation and updates, go to the WebSphere Application Server library page at URL:

```
http://www.ibm.com/software/websphere/appserv/zos_os390/library.html
```

HTML and PDF books are updated as needed.

Related documentation

This section includes information on frequently used related documentation. For additional related documentation, see the "Bibliography" on page J-1.

Java documentation

For Java for OS/390 documentation, go to URL:

```
http://www.ibm.com/servers/eserver/zseries/software/java/
```

For information on servlets, JavaServer Pages (JSPs), and other Java services, go to URL:

```
http://java.sun.com
```

Web server

To obtain the most current version of your Web server documentation and information updates, go to the library page for your particular Web server. (See "Required OS/390 Web server" on page 1-1 for a list of the library pages for the different Web servers that can be used with WebSphere Application Server Standard Edition Version 3.5.)

Support services

For information on support options and resources, see "APARs and service updates" on page 1-10.

Chapter 1. Planning for installation

Software requirements

Compatible OS/390 releases

WebSphere Application Server Standard Edition Version 3.5, hereafter referred to as the Application Server, runs on OS/390 Version 2 Releases 8, 9, and 10, as well as all current z/OS releases.

Required OS/390 Web server

The Application Server requires one of the following OS/390 Web servers:

Web server	Releases	For Web server documentation and updates, go to URL:
V5.3 IBM HTTP Server	z/OS and OS/390 Release 10	http://www.ibm.com/software/websphere/httpservers/doc53.html
V5.2 IBM HTTP Server	OS/390 Releases 8 and 9	http://www.ibm.com/software/websphere/httpservers/doc52.html

The Application Server is implemented as a Go Web Server API (GWAPI) that executes within an OS/390 Web server address space. It is recommended that you give the Web Server access to the BPX.SERVER profile in the FACILITY class defined within the SAF facility. The system subsequently requires that any DLLs that are loaded into an address space, which has been granted access to BPX.SERVER, be loaded from a dataset that has been marked program-controlled. If an attempt is made to load code from a non-program controlled library, the system will issue an abend.

A recent Web server PTF removed the requirement that the Web server address space be defined with UID=0. The Application Server does not require this level of processing at either the Version 3.02 or 3.5 product levels. Because application code executes in this address space, it is strongly recommended that you apply one of the following Web server PTFs and redefine the Web server's UID with less authority than UID=0:

- UQ49251, if you are using a Version 5.1 Web server
- UQ49253, if you are using a Version 5.2 Web server
- UQ49254, if you are using a Version 5.3 Web server

Required Software Development Kit

Version 3.5 of the Application Server requires the Software Development Kit (SDK) Version 1.3.

For the most current Java for OS/390 documentation, go to URL:

`http://www.ibm.com/servers/eserver/zseries/software/java/`

Notes:

1. When you apply service to the SDK, the program control bits will be lost. Therefore, you need to reissue the **extattr** command each time you apply service to reset these control bits.
2. The WebSphere Application Server Version 3.5 for MultiPlatforms currently supports JDK Version 1.2.2.and SDK Version 1.3. Both development kits are often referred to as Java 2 based JDKs. Applications that have been developed and deployed using JDK 1.2.2 are bytecode compatible with SDK 1.3. Therefore, Web applications that have been developed and deployed on WebSphere Application Server Version 3.5 for MultiPlatforms can be executed, unchanged, in the WebSphere Application Server for OS/390 Version 3.5 environment. You do not need to recompile an application when moving it across platforms; simply re-deploy the executable files such as .jar and .class files.
3. Version 3.02 of WebSphere Application Server for OS/390 required JDK 1.1.8. JDK 1.1.8 is referred to as a Java 1 based JDK. Java 1 is not fully upwardly compatible to Java 2. Therefore, applications that currently execute on a Java 1 based JDK that make use of APIs and facilities that have been removed or changed in Java 2 must be changed prior to deploying them on Version 3.5 of WebSphere Application Server for OS/390. Java 1 based applications that do not use APIs that have been changed in Java 2 remain bytecode compatible. If your current Java 1 based application, such as a Web application that is executing in Version 3.02, does not use APIs that are incompatible in Java 2, you can re-deploy the executable files into the Java 2 environment without recompiling them.

For more information about differences in JDK and SDK levels, see Appendix A, “Migrating from previous Versions of the Application Server”, on page A-1 and the following Web site:

`http://java.sun.com`

Required Java Servlet API levels

Version 3.5 of the Application Server supports Servlets that are compliant with the Java Servlet Specification Version 2.2. Version 3.02 of the Application Server supported servlets written to the Java Servlet Specification Version 2.1. The Version 2.2 specification is not upwardly compatible with the Version 2.1 specification. For more information on changes introduced with Version 2.2, see Appendix A, “Migrating from previous Versions of the Application Server”, on page A-1 and the following Web site:

`http://java.sun.com`

To accommodate migration from Version 3.02, Version 3.5 of the Application Server provides a servlet compatibility mode. When running in compatibility mode, the Application Server does not enforce inconsistent behavior or semantics that are introduced in Version 2.2 of the Servlet API Specification. This implies that servlets that currently execute in Version 3.02 do not need to be changed before being deployed in Version 3.5 as long as the Application Server is being run servlet compatibility mode (see “Including Web components in a Web application” on page 3-5 page 3-5). It is strongly recommended that installations migrating from Version 3.02 to Version 3.5 initially operate the Version 3.5 Application Server in compatibility mode.

WebSphere Application Server Version 3.5.2 for MultiPlatforms also supports Java Servlet specification Version 2.2, and includes a compatibility mode features. Servlets developed and tested on version 3.5, 3.5.1, or 3.5.2 (running in compatibility mode) of the WebSphere Application Server for MultiPlatforms should be able to be deployed unchanged to WebSphere Application Server for OS/390 running in compatibility mode, except where an explicit programming model restriction is noted. Similarly, servlets developed and tested on WebSphere Application Server Version 3.5.2 for MultiPlatforms running in compliance mode should be able to be deployed unchanged to WebSphere Application Server for OS/390 running in compliance mode except where an explicit restriction is noted. See Appendix D, “Programming Model Restrictions”, on page D-1 for a description of these restrictions.

Versions 1.1 and 1.2 of the Application Server provided support for servlets written to the Servlet Version 2.01 API Specification level. The Servlet Version 2.01 API Specification is not upwardly compatible to the Servlet Version 2.1 API Specification. Certain APIs have been changed or deprecated. Therefore, a servlet that uses any of these APIs must be updated to accommodate these changes. Servlets which do not utilize these APIs should be able to be deployed unchanged to this version of the Application Server. See Appendix A, “Migrating from previous Versions of the Application Server”, on page A-1 for more information on how to determine if changes are required to the servlets you are migrating.

Required JavaServer Pages levels

Version 3.5 of the Application Server introduces support for JavaServer Pages written to the JavaServer Pages Specification Version 1.1 level. It also continues to maintain support for JavaServer Pages written to the 0.91 and 1.0 specification levels.

The desired JavaServer Page specification level can be specified as part of a Web application definition. Therefore, Web applications that were previously deployed in Version 3.02 that contained JavaServer Pages compliant with the 0.91 or 1.0 specification level do not need to be changed as part of migrating to Version 3.

For more information on differences between the JavaServer Pages specification levels, see Appendix A, “Migrating from previous Versions of the Application Server”, on page A-1 and the following Web site:

`http://java.sun.com`

Note: Even though JSPs written to the Version 0.91 Specification level are supported in Version 3.5 of the Application Server, it is recommended that you start converting these JSPs to the 1.0 or 1.1 Specification level. JSPs written to the 0.91 Specification may not be supported by future versions of the WebSphere Application Server product.

XML Document Structure Services

The Application Server provides XML Document Structure Services which consist of an XML parser at the XML4J API 2.0.15 level, a document validator, and a document generator for server-side XML processing. These features let you leverage the power of XML, a tagging alternative to HTML, and enable you to use the IBM WebSphere Studio product in conjunction with the Application Server.

For more information about using XML, see URL:

`http://www.alphaworks.ibm.com/`

DB2 Requirements

Optionally, you can configure Version 3.5 of the Application Server to share HTTP session state data among multiple Application Server instances executing on the same or different OS/390 images. These OS/390 images can be members of the same Parallel Sysplex.

To provide this support, the Application Server requires one of the following hosted databases:

- DB2 Version 5 with PTF UQ49041applied
- DB2 UDB for OS/390 Version 6 with PTF UQ49039 applied
- DB2 UDB for OS/390 and z/OS Version 7

Note: Hereafter, all of these versions will simply be referred to as DB2.

It is necessary for the DB2 dynamic load libraries (DLLs) to be marked as program controlled in order for DB2 to function correctly with the Application Server.

It also requires that the Resource Recovery Manager Services and the OS/390 System Logger be configured to enable the JDBC driver to use the underlying Resource Recovery Services Attachment Facility (RRSAF). See one of the following publications for more information about configuring the Resource Recovery Manager Services Attachment Facility:

- *DB2 for OS/390 Version 5 Application Programming and SQL Guide* , SC26–8958
- *DB2 UDB for OS/390 V6 Application Programming and SQL Guide*, SC26–9004
- *DB2 UDB for OS/390 and z/OS V7 Application Programming and SQL Guide*, SC26-9933

Note: Alternate attach mechanisms such as the Call Attach Facility (CAF) are not supported in the Application Server environment.

See *OS/390 Parallel Sysplex Systems Management*, GC28–1861, for more information about the MVS System Logger.

See “Enabling communication with DB2” on page E-1, and “Session clustering” on page 5-3 for additional information about using DB2 with the Application Server.

Periodically, service updates are made to the JDBC driver. Please check the DB2 Preventive Service Planning (PSP) Bucket documentation on IBMLink for the most current information on APAR fixes and service updates. To access IBMLink on the Web, go to URL:

`http://www.ibm.com/ibmlink/`

Important Maintenance Note: When you apply service to DB2, the program control bits you set in order to enble DB2 to work correctly with the Application Server will be lost. Therefore, you need to reissue the **extattr** command each time you apply DB2 service.

OS/390 Workload Management considerations

In an OS/390 production environment, Workload Management (WLM) running in Goal mode can be used to balance workloads and distribute resources among competing workloads. To exploit the benefits of WLM, the Web server can be enabled for running in Scalable Server mode. This means that the Web server is configured for WLM support using the ApplEnv directive, and is started using the -SN (subsystem name) option.

For more information on WLM and Goal mode, refer to the *OS/390 MVS Planning: Workload Management* book. You can view this book on the Web at URL:

```
http://www.ibm.com/s390/os390/bkserv/
```

For more information on Scalable Server mode and enabling WLM support on the Web server, refer to your Web server documentation. For information on accessing Web server documentation, see "Required OS/390 Web server" on page 1-1.

System Authorization Facility (SAF) Support

Version 3.5 Application Servers can be configured to perform access control checks against Web resources. To provide this support, you must create resource profiles within the configured SAF product. See "Securing Web components" on page 3-19 for more information on access controls for Web resources.

Connector support

The Application Server supports Version 1.1 of the Common Connector Framework (CCF) architecture. CCF defines a framework which allows run-time providers, connector providers, and application code to collaborate in performing access to Enterprise Information Systems such as CICS and IMS. The Application Server implements the required infrastructure interfaces such that CCF Version 1.1 connectors can be deployed into its run-time environment. This implementation, which was contained in the **ccf.jar** file in previous versions of the Application Server, is now loaded by default into the Application Server's classpath at startup. Therefore, you no longer have to specifically enable CCF at run time.

The Visual Age for Java Enterprise Access Builder feature can generate the application code (command beans) that the connector needs to access the existing system. However, before deploying code generated by Visual Age for Java Version 3.x into the Application Server run-time environment, ensure that you have configured the necessary Visual Age for Java prerequisite support. This support, and corresponding installation documentation for installing it, is available at the following URL:

```
http://www.ibm.com/software/ad/vajava/
```

Also, ensure that the Visual Age for Java generated CCF code you are deploying has been generated from a Version 3.5x level of Visual Age for Java. Consult the Visual Age for Java documentation for information on considerations when moving to different levels of Visual Age for Java and/or when targeting different levels of the Application Server for deployment.

The IMS Connect product (5655-E51), which is a replacement for the IMS TCP/IP OTMA Connection (ITOC), must also be installed before IMS can communicate with the Application Server. This product can be downloaded from the following IMS product Web site:

http://www.ibm.com/software/data/ims/

For more information about setting up IMS and the Application Server to communicate with each other, see “Enabling communication with IMS using IMS Connect and IMS Connector for Java” on page E-6.

The CICS Connector is available via Version 4.0 of the CICS Transaction Gateway product (5648–B43). For more information about setting up CICS and the Application Server to communicate with each other, see “Enabling communication with CICS” on page E-5. For more information about this product and how to purchase it, go to URL:

http://www-4.ibm.com/software/ts/cics/platforms/desktop

As with the Visual Age for Java server-side support, you must obtain the necessary packages and define them to the Application Server classpath so that they can be accessed at run time by the requesting application. Connectors that are not compliant with the CCF Version 1.1 architecture will not execute in the Application Server Version 3.5 run-time environment. Therefore, connectors shipped with Visual Age for Java Version 2.0 are not guaranteed to execute in this environment.

IBM Distributed Debugger and Object Level Trace support

The IBM Distributed Debugger and Object Level Trace tools, which are shipped as part of the VisualAge for Java product, can be used to collect problem determination information about servlets and Version .91 and 1.0 level JSPs running in WebSphere Application Server V3.5 on OS/390. These tools provide graphical representation of the interactions between client/server applications, via the Object Level Trace Viewer on a workstation, thus enhancing a user’s ability to debug distributed applications.

Notes:

1. These tools can only be used with the Application Server for OS/390 V3.5 if the hosting Web server is running in standalone mode.
2. V1.1 JSPs running on an OS/390 V3.5 Application Server can not be debugged using these tools.

In order to work with a V3.5 Application Server, the VisualAge for Java product must be at a Version 3.5.3 level, or higher. This version of VisualAge for Java can be downloaded from the following Web site:

http://www7.software.ibm.com/vad.nsf

Note: You must be a registered member of the VisualAge Developer Domain before you can download this product. If you are not already registered, click on ″Register″ in the left hand margin of this Web page.

The IBM Distributed Debugger and Object Level Trace products have their own detailed user documentation that is available once you install VisualAge for Java Version 3.5.3 on a Windows NT, Windows 2000, or AIX platform. This documentation is also included in the WebSphere Application Server Information Center for Distributed Platforms, which is available at URL:

http://www-4.ibm.com/software/webservers/appserv/infocenter.html

Installing and running Object Level Trace

The README.txt file included with IBM Distributed Debugger and Object Level Trace provides general information on how to set up the tools to work with an

OS/390 system. You must also make the following changes to the Application Server was.conf file before you can use the IBM Distributed Debugger and the Object Level Trace tools with the OS/390 Application Server:

- Set the **objectleveltrace.enabled** property to **true**.
- Update the **objectleveltrace.host** property with the host name of the workstation where the IBM Distributed Debugger and Object Level Trace tools are installed.
- Update the **objectleveltrace.port** property with the the OLT Server TCP/IP port as it is defined for the IBM Distributed Debugger and Object Level Trace tools on your workstation. To determine this value:
  1. From the Object Level Trace Viewer window, select File/Preferences. This opens the Browser Preferences window.
  2. In the Browser Preferences window, highlight the OLT folder. Use the value that appears for ″OLT Server TCP/IP port″ as the value you specify on the **objectleveltrace.port** property.

See Appendix B, “was.conf file template”, on page B-1 for more information about these properties.

Once you have installed these tools, and updated the was.conf file properties use the following procedure whenever you want to run Object Level Trace:

1. Start the Object Level Trace Viewer on your workstation.
2. Set its Execution mode to ″Trace only″.
3. Start the Application Server on your OS/390 system.
4. Using a browser, request a servlet or JSP to be traced in the Application Server on OS/390.

Output from Object Level Trace should begin being displayed in the Object Level Trace Viewer window on your workstation. (See the IBM Distributed Debugger and Object Level Trace documentation for information about the content of this output.)

Installing and Running the IBM Distributed Debugger

Before installing the IBM Distributed Debugger on your OS/390 system, you must:

- Make sure that IBM SDK 1.3 is installed on your z/OS or OS/390system and is at Service Release level 7 with PTF UQ53763 installed.
- Create the following directory structures, if they do not already exist:

  ```
  /usr
  /usr/lpp
  /usr/lpp/IBMDebug
  /usr/lpp/IBMDebug/bin
  /usr/lpp/IBMDebug/lib
  ```
- In binary mode, FTP file **derdebug.jar** from the **/extras/390/lib/** directory on your workstation to the **/usr/lpp/IBMDebug/lib** directory on the OS/390 system.
- In binary mode, FTP file **irmtdbgj** from the **/extras/390/bin/** directory on your workstation to the **/usr/lpp/IBMDebug/bin** directory on the OS/390 system.
- Issue the following command to make the Distributed Debugger executable:

  ```
  chmod +x irmtdbgj
  ```
- Make the following updates to your HTTP Server’s httpd.envvars file:
  – Prepend **/usr/lpp/IBMDebug/bin** to the PATH statement.
  – Prepend **/usr/lpp/IBMDebug/lib** to the LIBPATH statement

- Make the following changes to the Application Server was.conf file, unless you have already made them to enable the Object Level Trace tool:
  - Set the **objectleveltrace.enabled** property to **true**.
  - Update the **objectleveltrace.host** property with the host name of the workstation where the IBM Distributed Debugger and Object Level Trace tools are installed.
  - Update the **objectleveltrace.port** property with the the OLT Server TCP/IP port as it is defined for the IBM Distributed Debugger and Object Level Trace tools on your workstation.
- Change following properties in the Application Server default_global.properties file:
  - Set the **appserver.product.java.jvmdebug.debug** property to **true** to enable the IBM Distributed Debugger support.
  - Update the **appserver.product.java.jvmdebug.port** property with the number of the pre-selected port used as the JVM remote debugger port. For more information on pre-selected port, see the section ″Java 2 JDK-related issues″ in the README.txt file that is installed on your system when you install the IBM Distributed Debugger and the Object Level Trace tools.

  Note: When the **appserver.product.java.jvmdebug.debug** property is set to **true**, the **appserver.product.java.jvmconfig.jit** property is automatically set to **NONE**, and any value specified in the default_global.properties file will be ignored.
- It is recommended that you also change the values specified for the following properties in the Application Server default_global.properties file to the indicated values:
  - Set the value on the **appserver.product.java.jvmconfig.mx** property to **512m**
  - Set the value on the **appserver.product.java.jvmconfig.ms** property to **512m**
  - Set the value on the **appserver.product.java.jvmconfig.oss** property to **512k**
  - Set the value on the **appserver.product.java.jvmconfig.ss** property to **512k**

See Appendix B, “was.conf file template”, on page B-1 and Appendix C, “default_global.properties file”, on page C-1 for more information about these Application Server properties.

Once you have performed all of the installation steps, use the following procedure whenever you want to run the Distributed Debugger:

1. Start the Object Level Trace Viewer on your workstation.
2. Set its Execution mode to ″Trace and Debug only″.
3. Check the Step-by-step Debugging Mode selection under the Options pull-down menu.
4. Start the Application Server on your OS/390 system.
5. Using a browser, request the servlet or JSP to be debugged in the Application Server on OS/390.
6. If you are debugging a V.91 JSP,
   a. When the ″Source File Name″ dialog box appears in the Object Level Trace Viewer requesting the location of the javax/servlet/http/HttpServlet.java source code, select ″Cancel″.
   b. Then from the ″Debug menu″ pull-down list select ″Step Over″ and then ″Step Over″ again, and then ″Step into″ to get to the JSP .91 source code you want to debug.

7. When the ″Method BreakPoints″ dialog appears in a window on your Workstation, indicating that the servlet or JSP class is loaded, select the only entry and select ″OK″.

The IBM Distributed Debugger and Object Level Trace documentation describes, in detail, the information that will start appearing in Object Level Trace Viewer window.

Application development tooling considerations

WebSphere Application Server Version 3.5 is designed to support Web-based application components (servlets, JSPs, JavaBeans, etc.) which are generated by the IBM VisualAge for Java Version 3.5 application development environment, and Web components that are generated and/or published by WebSphere Studio Version 3.5.

The version number of the Visual Age for Java and WebSphere Studio products indicates the WebSphere Application Server run-time environment which they are intended to support. For example, Visual Age for Java Version 3.5x is intended to be compatible with WebSphere Application Server for OS/390 Version 3.5 run-time environments. In cases where extended support is provided by the tooling (i.e., the ability to operate with multiple versions of Application Server run-time environments), this capability will be documented in the Visual Age for Java and WebSphere Studio product documentation.

Deploying Components generated by VisualAge for Java

The VisualAge for Java, Enterprise Edition, Version 3.5 product for Windows NT that ships on CDs includes the Enterprise Toolkit for OS/390. ET/390 provides a way to write Java programs at the workstation, and then export these programs to run in a remote OS/390 Java Virtual Machine (JVM). The documentation that is shipped with the workstation version of ET/390 and the host version (5655JAV01) is identical.

Before you can deploy components generated by VisualAge for Java into the Application Server run-time environments, you must have the following JAR files installed on your file system and their location must be included on the **appserver.classpath** property, which specifies the directories in your JVM classpath:

- ivjdab.jar
- recjava.jar
- eablib.jar

These files can be found in the **\extras\runtime30** directory on Disk 2 of the VisualAge for Java, Enterprise Edition, Version 3.5 product CDs, or in your temporary directory where you extracted IDE 4 of 9 (Deployment Environments and online documentation in PDF format), if you have an electronic version of VisualAge for Java.

In addition, if you are deploying components, such as a command bean generated by EAB, that require Common Connector Framework (CCF) support, you must also include the necessary connector files. For example, if you are using the CICS Transaction Gateway product to provide your connector support, you must also include the following JAR files on the **appserver.classpath** property:

- ctgclient.jar
- ctgserver.jar

These files are located in the CTG User Classpath that was set up during the CTG configuration process.

See Appendix B, "was.conf file template", on page B-1 for more information about the **appserver.classpath** property.

For more information about IBM Visual Age for Java, see the product Web site, at URL:

`http://www.ibm.com/software/ad/vajava/`

Deploying Components generated by WebSphere Studio

The Application Server does not require any supporting server side classes to be installed in order to deploy components that are generated and/or published by WebSphere Studio.

The IBM WebSphere Studio combines easy-to-use wizards with site design and Java development tools, simplifying and speeding the application development process. WebSphere Studio is available on Windows NT, Windows 2000, Windows 98, and Windows 95. See the *WebSphere Studio Guide* for a detailed description of this product.

For the most current information and to download a trial version of this product, go to the Studio Web site at URL:

`http://www.ibm.com/software/websphere/studio/`

Notes:

1. Before attempting to run servlets generated using the WebSphere Studio product, you must download the webtlsrn.jar file from WebSphere Studio to your OS/390 system and place it in the classpath of the Web applications that will be using these servlets. This file contains the WebSphere Studio utilities required to run the servlets generated using the WebSphere Studio product.
2. A copy of the databeans.jar file is shipped as part of the Application Server product. **DO NOT** replace this copy of the databeans.jar file with the copy that is shipped with the WebSphere Studio product.

Installation and configuration process changes

As with Version 3.02, for Version 3.5, the Application Server properties are contained in a single configuration file. See Appendix A, "Migrating from previous Versions of the Application Server", on page A-1 for information about properties that have been added or changed for this version of the Application Server. See Appendix B, "was.conf file template", on page B-1 for a copy of the template for this configuration file.

APARs and service updates

For the most current information on APAR fixes and service updates, check the *WebSphere Application Server for OS/390 Version 3.5 Program Directory*, which is shipped with the product, and the WASSE350 Preventive Service Planning (PSP) bucket. You can link to the most current copy of the Program Directory and the PSP bucket from the Application Server library page at URL:

`http://www.ibm.com/software/websphere/appserv/zos_os390/library.html`

The PSP Bucket is also available on IBMLink. To access IBMLink on the Web, go to URL:

```
http://www.ibm.com/ibmlink/
```

Chapter 2. Installing and customizing the Application Server

Installing the Application Server

Instructions for installing the WebSphere Application Server Standard Edition element of WebSphere Application Server for OS/390 Version 3.5 can be found in the WebSphere Application Server for OS/390 Version 3.5 Program Directory. This program directory is shipped with the product and can also be obtained from the product library page at URL:

```
http://www.ibm.com/software/websphere/appserv/zos_os390/library.html
```

After the installation process has been completed, the resulting HFS containing the install-image can be mounted to any execution system that is to host an Application Server. The name of the root directory where the install-image of an individual execution system is located is referred to as the *applicationserver_root*. The default *applicationserver_root* is **/usr/lpp/WebSphere**.

Note: An install-image may be mounted as read-only on the execution systems on which it is going to execute. An Application Server, which will be executing within a Web server address space under the identity of that address space, will require read access to the *applicationserver_root* at run time. Therefore, the mounted image **must** have the appropriate file permissions to allow read access.

Verifying the Application Server installation

The Application Server installation process can be verified by:

1. Configuring a Web server to host an Application Server.
2. Starting the Application Server
3. Invoking the Installation Verification Program.

Configuring a Web server to host an Application Server

To run an Application Server within a Web server address space, you must:

1. Add the following Web server directives to the httpd.conf configuration file of any Web server that will be hosting an Application Server to provide the Web server with the entry point to the Application Server's initialization, request processing, and exit routines. These routines exist as entry points init_exit, service_exit, and term_exit, respectively, within the was350plugin.so DLL. The was350plugin.so DLL is found within the *applicationserver_root*/**AppServer/bin** directory.

```
ServerInit applicationserver_root/AppServer/bin/
   was350plugin.so:init_exit applicationserver_root
Service /webapp/examples/* applicationserver_root/AppServer/bin/
   was350plugin.so:service_exit
ServerTerm applicationserver_root/AppServer/bin/was350plugin.so:term_exit
```

Notes:

a. In this example, the ServerInit and Service directives are split for printing purposes. In the actual httpd.conf file, each directive is on a single line.

b. If you do not want to use the default was.conf file to configure the Application Server, you must specify the name of the was.conf file you want to use at the end of the ServerInit directive. See"Configuring the Application Server" on page 2-5 for an example where the name of a was.conf file was added to the ServerInit directive.

c. The Web server interprets a blank in a directive specification as a delimiter and a number sign (#) as the beginning of a comment that should be ignored. Therefore, if you need to use a blank or number sign in a directive, you **must** include a backslash (\) before the blank or number sign to enable the Web server to correctly process the directive.

d. If a servlet sets an HTTP response code by any means, such as using methods lastModified() or setStatus(), and the client does not receive the expected response code, add the following directive to the HTTP Server configuration file:

```
ServiceSync On
```

2. Make sure that the JAVA_HOME environment variable contained in the hosting Web server's envvars file points to the exact location where the required level of the Software Development Kit (SDK) is installed on your system. (See "Required Software Development Kit" on page 1-2 for the Development Kit level that is required by the Version 3.5 Application Server.)

3. Append the Application Server message catalog directory, *applicationserver_root*/AppServer/msg/%L/%N, to the existing NLSPATH statement specified in the HTTP server's envvars file. For example, if NLSPATH was set as:

```
/usr/lib/nls/msg/%L/%N:/usr/lpp/internet/%L/%N
```

and the Application Server is installed in **/usr/lpp/WebSphere**, change the NLSPATH to:

```
/usr/lib/nls/msg/%L/%N:/usr/lpp/internet/%L/%N:/usr/lpp/WebSphere/AppServer
    /msg/%L/%N
```

The Application Server environment is a Java Server Runtime environment capable of hosting Web components such as Java Servlets and JavaServer Pages. The environment is structured to accommodate inclusion of other services and facilities (such as JDBC support) for use by the Web components. Many of these Java services are implemented assuming that only Java compliant implementations of these services are to exist within the Java based run-time environment. Many packages are not guaranteed to work properly when native code in other versions of the services are introduced into this environment. Other packages impose a large set of restrictions that must be adhered to.

For example, the JDBC driver provided by DB2 makes use of the RRSAF package when connecting to DB2. DB2 explicitly documents that only one type of attach package can exist within a given address space. Therefore, any native code that is configured in the address space, such as another GWAPI plug-in that makes use of the Call Attach Facility (CAF), is not allowed. For cases where another GWAPI program makes use of RRSAF natively to attach to DB2, the program must consider whether its management of this connection conflicts with the assumptions made by the JDBC driver (i.e., state management on threads within this multi-threaded execution environment) within its implementation.

You must be sure to accommodate the use of JDBC by other system components. An attempt to reconcile any deltas by simply not including application code that makes use of JDBC in the same address space with code that uses other native facilities may not be sufficient because the Application Server often makes uses of JDBC in its implementation of persistent HTTP Session state.

Configuring multiple instances of the Application Server or multiple product levels of the Application Server within the same address space is not permitted. Therefore, when updating an existing httpd.conf file that contains existing Application Server directives, you **must replace** the existing ServerInit, ServerTerm, and Service directives with corresponding directives containing the new format previously described in this section.

Notes:

1. To minimize the risk of encountering an incompatibility in system services, IBM recommends that you do not run other plug-in routines that use different programming model facilities in the same address space where you are running the Application Server.
2. As a result of the install process, the DLL libraries for the Application Server contained in the *applicationserver_root*/**AppServer/bin** directory are marked as program controlled. You must be sure to preserve these settings.

Starting the Application Server

At execution time, an Application Server will run within any Web Server address space in which an Application Server has been properly configured. Both the Web server and the Application Server issue status messages to the standard error (stderr) log file to indicate their progress throughout their initialization process.

The following depicts the content of the stderr log file upon successful initialization of the Application Server. Successful completion of initialization is indicated by messages in the log file from both the Application Server and the hosting Web server indicating that the server is ready.

```
........... This is IBM HTTP Server web_server_type
........... Built on date at time.
........... Started at day date time year.
........... Running as "your_server_name", UID:OS/390_UNIX_UID,
                GID:OS/390_group_ID.
IMW0234I Starting.. httpd

This is IBM WebSphere Application Server for OS/390 3.50 built on OS/390 Version 2
   Release release_number,WAS Service Level service_level_number

Built on date at time.
Started at day date time year

Started Queue State = HTTPD

Started Server Type = STANDALONE
WAS Startup Parameter -- Install Root = /applicationserver_root
WAS Startup Parameter -- Configuration file = applicationserver_root/AppServer/
   properties/was.conf
WAS Startup Parameter -- Bootstrap file = applicationserver_root/AppServer/
   properties/default_global.properties
WAS Startup Parameter -- JDK install directory (JAVA_HOME) = /usr/lpp/java/
```

```
   J1.3.0/IBM/J1.3

WAS Startup Parameter -- Plugin Logging Level = WARNING

WAS Startup Parameter -- Plugin Logging Directory = /your_server_name/logs
WAS generated CLASSPATH follows:
CLASSPATH entry: applicationserver_root/AppServer/lib
CLASSPATH entry: applicationserver_root/AppServer/lib/ant.jar
CLASSPATH entry: applicationserver_root/AppServer/lib/antxalan_1_1.jar
CLASSPATH entry: applicationserver_root/AppServer/lib/bsf.jar
CLASSPATH entry: applicationserver_root/AppServer/lib/databeans.jar
CLASSPATH entry: applicationserver_root/AppServer/lib/dertrjrt.jar
CLASSPATH entry: applicationserver_root/AppServer/lib/ibmant.jar
CLASSPATH entry: applicationserver_root/AppServer/lib/ibmwebas.jar
CLASSPATH entry: applicationserver_root/AppServer/lib/jsp10.jar
CLASSPATH entry: applicationserver_root/AppServer/lib/lotusxsl.jar
CLASSPATH entry: applicationserver_root/AppServer/lib/servlet.jar
CLASSPATH entry: applicationserver_root/AppServer/lib/xerces.jar
CLASSPATH entry: applicationserver_root/AppServer/lib/xml4j.jar
CLASSPATH entry: applicationserver_root/AppServer/properties
CLASSPATH entry: applicationserver_root/AppServer/classes
CLASSPATH entry: /usr/lpp/java/J1.3.0/J1.3/lib/tools.jar
CLASSPATH entry: /usr/lpp/java/J1.3.0/J1.3/lib/ext/RACF.jar
............  End of generated CLASSPATH

IMW0234I Starting.. httpd
............ WAS Startup Parameter -- Servlet Engine Logging Level = WARNING
............ WAS Startup Parameter -- Servlet Engine Logging Directory =
   /your_server_name/logs
............ WAS Startup Parameter -- Servlet Engine Working Directory =
   /your_server_name/work

............ IBM WebSphere Application Server native plugin initialization
                 went OK :-)
IMW0235I Server is ready.
```

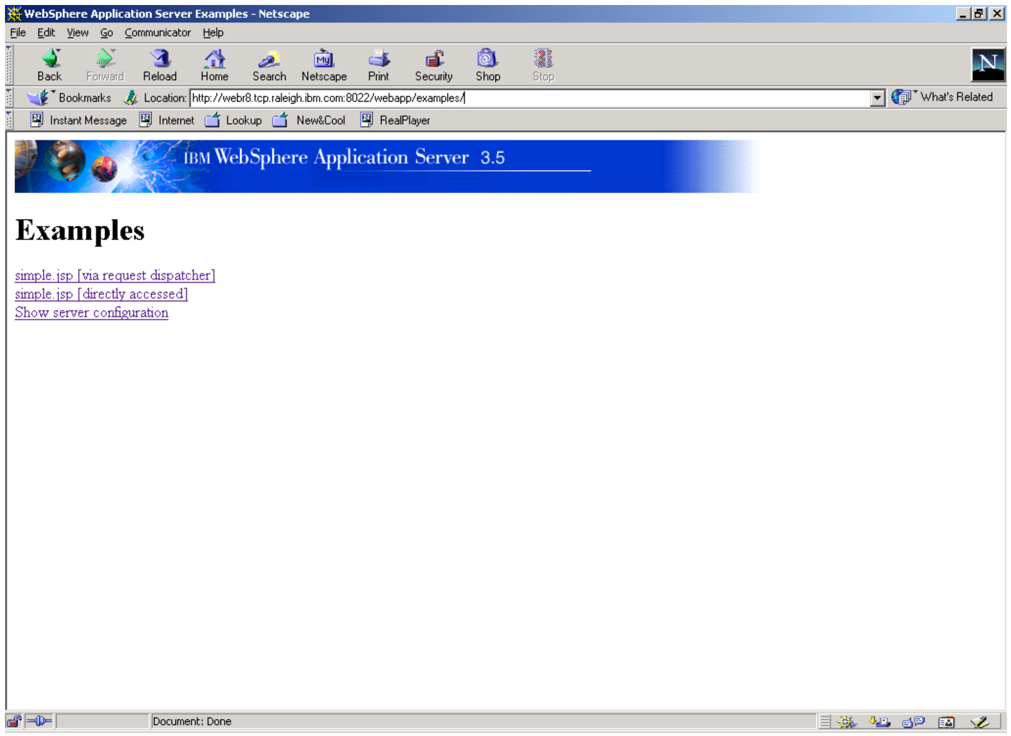
Note: The IBM Developer Kit for the Java Platform, used with previous versions of the Application Server, required that system classes (classes.zip) had to be added to the Application Server CLASSPATH. SDK 1.3, used withVersion 3.5 of the Application Server, does not have this requirement.

Message IMW0235I indicates that your Web server has successfully initialized. The preceding ″smiley face″ message indicates that the Application Server successfully initialized. It is possible to get message IMW0235I without the preceding ″smiley face″ message if the Application Server did not successfully initialize. If you do not receive message IMW0235I, an error has occurred during the Web server initialization process.

Invoking the Installation Verification Program

Once the Application Server is started, you can verify the install by entering the following URL from a browser to invoke the IBM-provided Installation Verification Program:

```
http://your.server.name/webapp/examples/index.html
```


WebSphere Application Server Examples - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://webr8.tcp.raleigh.ibm.com:8022/webapp/examples/
What's Related
Instant Message Internet Lookup New&Cool RealPlayer
IBM WebSphere Application Server 3.5

Examples

simple.jsp [via request dispatcher]
simple.jsp [directly accessed]
Show server configuration

Document: Done

You can then select one of the examples from this page and verify that the Application Server has properly installed.

- The Show Server Configuration example will display your current Application Server configuration settings.
- Either of the simple.jsp examples will verify that level 1.1 JSPs work properly in your Application Server environment.

Note: If you have changed any of the default settings in the was.conf file for this instance of the Application Server, make sure the **session.enable** property is set to **true** before running the Installation Verification Program.

Configuring the Application Server

As with Version 3.02 of the Application Server, Version 3.5 can be configured by updating properties provided in a was.conf configuration file. The Application Server takes as input the fully qualified name of a was.conf file. The properties within this file are each specified on a single line and can set things such as a classpath or document root.

If you are migrating from Version 1.1 or 1.2 of the Application Server, you are not required to deviate from your current *server_model_root* structure. The properties within the new was.conf file enable you to configure the Application Server so that it uses your existing directory structures and specifications. See Appendix A, “Migrating from previous Versions of the Application Server”, on page A-1 for more migration information.

Using a was.conf file to set configuration properties

The fully qualified location of the Application Server’s configuration file (was.conf) can be specified within the Web Server’s configuration file as the second positional parameter to the Application Server initialization routine. When a value is not supplied for this parameter, the Application Server uses the default was.conf file located at *applicationserver_root***/AppServer/properties/was.conf**. This default file is designed to support execution of the Installation Verification Program (see "Invoking the Installation Verification Program" on page 2-4).

The property values specified in the was.conf file determine how the Application Server handles Web applications deployed within it. You can use the was.conf template that is provided with the product to create a was.conf file for a particular instance of the Application Server. The template is located in the *applicationserver_root***/AppServer/properties/was.conf.template** file; a copy is also provided in Appendix B, "was.conf file template", on page B-1.

You must add a ServerInit directive to the hosting Web server's httpd.conf file that specifies the Application Server's location (*applicationserver_root*) and the name of the associated was.conf file. The following example shows the ServerInit directive that was added to the hosting Web server's httpd.conf file for an Application Server with an *applicationserver_root* of **/MyExecutionSystem/usr/lpp/WebSphere**, and whose configuration is provided in the was.conf file **/MyExecutionSystem/usr/ServerDefs/PayrollServer.conf**:

```
ServerInit /MyExecutionSystem/usr/lpp/WebSphere/AppServer/bin/
   was350plugin.so:init_exit/MyExecutionSystem/usr/lpp/WebSphere,/
   MyExecutionSystem/usr/ServerDefs/PayrollServer.conf
```

Note: In this example, the ServerInit directive is split for printing purposes. In the actual file, this directive is on a single line.

| If no name is specified, the default was.conf file,
| *applicationserver_root***/AppServer/properties/was.conf**, is used.

During initialization, the Application Server performs syntax checking on the values within the was.conf file. If an error is detected, initialization will fail and appropriate error messages will be sent to the stderr log.

Specifying configuration properties

The following types of properties can be included in your was.conf file:

- **Application Server run-time properties:** You can provide properties within the was.conf file that influence the behavior of the Application Server's run-time code, and apply to all Web applications deployed in the Application Server . Among other things, you can specify:
  - Information that is to be provided to the Java Virtual Machine that is to be instantiated
  - The level of logging that is desired
  - The physical directories to use for logging output.
  - The physical directories to use for temporary files created by the Application Server, such as classes created during JSP processing.

  Properties in this category have the format of **appserver.***property_name***=***value*. For more information on configuring the Application Server run-time properties, see "Customizing the Application Server" on page 2-8.
- **Virtual Hosts:** Virtual hosts can be explicitly configured to the Application Server. Properties relating to the configuration of virtual hosts have the format **host.***host_name.property_name***=***value*. For more information on configuring virtual hosts, see "Configuring a virtual host" on page 3-2.
- **Web applications:** If a request routed to the Application Server is to be satisfied, it must ultimately be routed to a Web application which satisfies the request.

Web applications provide the ability to group a set of Servlets, JSPs, and static files together so they can be managed as a single unit. Web applications can be configured within a defined virtual host. Properties relating to the configuration of Web applications have the format **webapp.***webapp_name.property***=***value*. Properties relating to the deployment of Web applications have the format **deployedwebapp.***webapp_name.property***=***value*For more information on defining and deploying Web applications, see “Defining and deploying Web applications” on page 3-3.

- **JDBC Database Connection Pools:** You can use properties having the format **jdbcconnpool.***pool_name.property_name***=***value* to define database resources for your applications to use. For more information on configuring database connection pools, see Chapter 4, “Accessing relational databases”, on page 4-1.
- **HTTP Session Support:** The Application Server provides support for the HTTP Session API defined by the Servlet API Specification. Properties relating to the configuration of HTTP Session support have the format of **session.***property_name***=***value*. For more information on configuring HTTP Session support, see Chapter 5, “Session tracking”, on page 5-1.

Appendix B, “was.conf file template”, on page B-1 contains detailed descriptions of all configuration properties contained in the was.conf file.

Updating Application Server properties

When the Application Server initialization routine executes, it creates an Application Server instance within the process of the hosting Web server. The Application Server instance is created using the values provided to the initialization routine through the associated was.conf file. The Application Server instance that is created remains resident for the life of the Web server process in which it was created. Therefore, before the Application Server can recognize changes in its configuration, it must be instantiated within a new physical address space.

When the Application Server is executing in a Web server that is running in stand-alone mode, the Web server must be stopped and a new instance started before the Application Server recognizes changes in its configuration. Because a restart of the Web Server running in stand-alone mode does not destroy and recreate the Web server process, configuration changes to the Application Server are not recognized after a restart. The Application Server will continue to execute using the configuration settings specified when the process was originally created.

If the hosting Web server is running in scalable server mode, it is recommended that you configure the Application Server to execute within a queue server. Scalable server mode allows new queue servers to be started to replace existing processes. The Application Server instances that will be started within the new queue servers will read and process any changes that exist in the configuration file. While the queue server address spaces are being recycled, the Queue Manager representing the network endpoint remains active. This enables you to make changes to the Application Server’s configuration without clients experiencing an interruption in service.

Directing requests to the Application Server

The Service directives within the hosting Web server’s httpd.conf file indicate which requests the Application Server is to process. If the requested URL matches a URL or URL pattern specified in a directive in the httpd.conf file, that request is routed to the Application Server. The Application Server then uses webapp definition statements to determine which Web Application the request should be routed to.

The **deployedwebapp.** *<webapp-name>***.rooturi=** properties in the was.conf file for an Application Server instance defines what URLs the Application Server is able to process for a particular Web application. When a request is routed to a Web application, the Web application attempts to satisfy it using the physical resources that are defined within its configuration. If the request is unable to be satisfied, it will be rejected. (See “Mapping URLs to Web components” on page 3-17 for information on how to define a URL mapping in the was.conf file.)

The hosting Web server performs protection directive processing prior to invoking the Application Server service routine. Before passing a request to the Application Server Service routine, the Web server selects a thread in which to dispatch the request and establishes any necessary security credentials on that execution thread. The Application Server then attempts to process requests synchronously on the HTTP execution thread in which its service routine is invoked. In cases where this is not possible, the Application Server is responsible for propagating the necessary context (i.e. execution identity, etc.) to any subsidiary threads prior to invoking the target application component such as a servlet or a JSP.

Note: The Application Server does not provide facilities to initiate client authentication. Instead, it relies on the security facilities provided by the hosting Web server. The Application Server is able to subsequently perform access control checks using the identity that is assigned during the Web server’s authentication processing.

Customizing the Application Server

Using the run-time properties in the Application Server was.conf file, you can :

- Specify the fully qualified name of a file containing the properties that are to be used to instantiate a Java Virtual Machine (JVM).
- Maintain compatibility with existing applications while simultaneously supporting the Java Servlet API 2.2 specification.
- Specify that you want to use the copy of the Xerces.jar and Xalan.jar files distributed with Apache rather than the version that is shipped with the Application Server.
- Specify the logging level for customer-directed messages.
- Specify a working directory that the Application Server can use to create a temporary file.

Specifying the name of a file containing the properties for instantiating a JVM

The **appserver.jvmpropertiesfile** property is used to specify the fully qualified name of a file containing the properties that are to be used to instantiate a Java Virtual Machine (JVM) for use by the Application Server. If a value is not specified for this property, the Application Server defaults to using the *applicationserver_root***/AppServer/properties/default_global.properties** file. This file contains reasonable defaults for executing the JVM within the Application Server at the supported levels. IBM recommends that you continue to use this default file until individual observation identifies an explicit need to change it. Using the default values ensures that you will get the latest recommended defaults as they are applied through service updates. The configuration settings that can be specified in a JVM properties file are described in detail within the default file. See Appendix C, “default_global.properties file”, on page C-1 for a copy of this file.

Maintaining compatibility with existing applications

The **appserver.compliance.mode** property is used to maintain compatibility with existing applications that you may have been running on a Version 3.02 Application Server, while simultaneously supporting the Java Servlet API 2.2 specification. To ensure compatibility, this new property enables you to indicate to the Application Server whether the Web applications you are running comply with the Java Servlet API 2.1 or 2.2 specification. If the components in a Web application comply with the Java Servlet API 2.1 specification, you specify **false** to indicate to the Application Server that you want it to run in compatibility mode; if they comply with the Java Servlet API 2.2 specification, you specify **true** for this property. **false** is the default value for this property.

The following table describes how the **appserver.compliance.mode** property setting affects Servlet API classes and methods, and various application functions. In this table, ″Compatibility Mode″ indicates that the applications comply with the Java Servlet API 2.1 specification, and ″Compliance Mode″ indicates that the applications comply with the Java Servlet API 2.2 specification. Make sure you understand all of the application processing implications noted in this table before changing the setting of this property to **true**.

Method or function	**Compatibility mode**	**Compliance mode**
Error-page tags in the .webapp file	The >error-page< contains a string that is the relative path to the Web application’s default error page.	The >error-page< contains the following tags: • >location< • >exception-type< • >error-code< These tags are only available in a .webapp file. Since there is no corresponding property in the was.conf file, this function is only available when a .webapp file is used to define a Web application.
getCharacter Encod ing() method	If the client request did not send any character encoding data, the default encoding of the server JVM is returned.	If the client request did not send any character encoding data, **null** is returned.
Default content type on *response buffer reset*	On *response buffer reset*, the content type of the request is reset to **text/html**.	On *response buffer reset*, the content type is cleared and not set to a default value.
getMimeType() method	If the file extension does not map to a valid mime type, the mime type **www/unknown** is returned.	If the file extension does not map to a valid mime type, **null** is returned.
HTTP Session scoping	Values placed in the HTTP Session object have a **global scope**, across all Web applications.	Values placed in the HTTP Session object have a **scope limited to the Web application** that created the value.

Method or function	Compatibility mode	Compliance mode
Request mapping behavior	• Exact mapping is not supported. Any URL pattern that is to be accepted and used as is, must end with with /* . • Wildcard mapping is an implied wildcard. Any URL pattern specified without /* on the end is assumed to be a wildcard rule, and /* is added in the servlet run-time environment. For example, /Servlet would be interpretted as /Servlet/*.	• The servlet specification pattern mapping logic is followed. This includes support for exact matches. • To specify a URL, the Servlet 2.2 specification allows the following syntax: 1. A string beginning with / and ending with /* specifies a wildcard match. 2. A string beginning with *. specifies an extension mapping. 3. All other strings are used as exact matches. • The Servlet 2.2 specification indicates how requests for resources are mapped to the appropriate resources. Mapping occurs in the following order: 1. exact match 2. longest wildcard match 3. matching extension 4. default servlet (defined by / URL)

Configuring the Application Server to use the Xerces.jar and Xalan.jar files distributed with Apache

If you prefer to use the copy of the Xerces.jar and Xalan.jar files distributed with Apache rather than using the version that is shipped with the Application Server, add the following property to the was.conf file:

```
appserver.java.system.property=com.ibm.servlet.classloader.delegate=false
```

When you add this property to the was.conf file, the Application Server will use the WebApp classloader to load classes before it uses the system classloader. You can then place the versions of the Xerces.jar and Xalan.jar files you want to use in the WebApp classpath and the Application Server will load these versions instead of the versions shipped with the Application Server product that reside in the system classpath.

If, at a later time, you decide to use the version of the Xerces.jar and Xalan.jar files shipped with the Application Server, simply change the value specified on this property to true. This will cause the system classloader to be used to load classes ahead of the WebApp classloader.

Specifying the logging level for customer directed messages

The **appserver.loglevel** property is used to select one of three levels of logging for customer-directed messages. The values that can be specified for this property are **ERROR**, **WARNING**, and **INFO**.

- Specifying **ERROR** instructs the Application Server to log only events that are fatal to the operation of the Application Server. This level represents the most minimal First Failure Data Capture.
- Specifying **WARNING** results in the logging of ERROR level messages as well as WARNING messages, which indicate a potential error. WARNING is the default value and the value recommended by IBM for normal operations.

- Specifying **INFO** results in extensive logging. **INFO** level logging is only recommended when you need to perform explicit problem determination.

Note: The Servlet 2.1 and Servlet 2.2 specifications define a log method on the ServletContext object that allows servlets to write messages to a destination provided by the underlying servlet engine implementation. The Application Server's implementation of log method will direct these messages to the destination log defined on the **appserver.logdirectory** property in the was.conf file. The Application Server treats all messages written through this method as information messages. Therefore, these messages will only be written to the destination log if the **appserver.loglevel** property in the was.conf file is set to INFO.

Specifying the log file directory

The **appserver.logdirectory** property is used to specify a directory in which the Application Server is to create log files at run time. This directory must exist before the Application Server is started.

At run time, the Application Server attempts to access this directory using the identity of the hosting Web server's process. Therefore, the Application Server requires permission to read or write to any files or directories in this path at run time.

During its initialization process, the Application Server uses the value provided for this property to create two files per Application Server instance within the specified directory structure. These files are then used throughout the life of the Application Server instance. The names for these two files will have the format ncf.log.*date.WLMmode.PID* and native.log.*date.WLMmode.PID*, where *date* is the date the log is created, *WLMmode* provides information about the hosting Web server's address space, and *PID* is the process ID of the hosting Web server's address space. The native.log file will be used to log messages produced by the Application Server's C code before entering Java; the ncf.log file will be used to log any System.out and System.err prints.

If the Web server is running in stand-alone mode, a value of **STANDALONE** is used for *WLMmode*. If the Web server is executing in scalable server mode and the Application Server is executing in a queue server address space, a value equal to the APPLENV name of that queue server is used for *WLMmode*. If the Web server is executing in scalable server mode and the Application Server has been configured to execute in the Queue Manager address space, a value of **QM** is used for *WLMmode*. IBM does not recommend this latter configuration.

If no value is specified for the **log directory** property, the output location is determined by the mode in which the hosting Web server is started. If the Web server is started from the Unix System Services shell, output is directed to STDOUT and STDERR. If the Web server is running as an OS/390 Started Task, output is directed to the location represented by the SYSPRINT DD within the JCL of that started task. If the Application Server needs to log messages for use by IBM Service personnel, it will do so within these same output destinations.

Specifying a working directory

The **appserver.workingdirectory** property is used to specify a working directory that the Application Server can use to create temporary files. If you do not specify a fully qualified path for this directory, the Application Server will use the default directory path and append the Application Server name to that directory path. (See Appendix B, “was.conf file template”, on page B-1 for more information about this property.)

Among other things, this directory is used to contain the output from Java Server Page compilations. This directory must exist before the Application Server is started. At run time, the Application Server will attempt to access this directory using the identity of the hosting Web server’s process. Therefore, the Application Server requires permission to read or write to any files or directories in this path at run time.

Note: Some of the files the Application Server creates under the working directory use fully qualified file names to reference other files in the working directory. Therefore, if you need to change the mount point of the working directory, you must first stop the Application Server and delete the contents of the working directory, including all of its subdirectories. You can then change the mount point and update the Application Server’s working directory property in the was.conf file. When the Application Server is stopped and then started again, it will recreate the directories and files within those directories as necessary.

Chapter 3. Defining virtual hosts and Web applications

Defining virtual hosts

Version 3.5 of the Application Server enables you to define virtual hosts. This support allows you to have multiple logical hosts which share the same IP address on a Web server.

A virtual host can have one or more DNS aliases. A DNS alias consists of a TCP/IP hostname and port number. A client request for a Web application, servlet, or related resource contains a DNS alias plus a Web path name that is unique to that resource.

Application Server virtual host support allows multiple Application Server hosts to be defined within a single Web server. When the Application Server is configured, at least one virtual host (the default virtual host that is provided with the Application Server) is associated with it. “Configuring a virtual host” on page 3-2 describes how to define additional virtual hosts if they are needed.

One or more Web Applications can be deployed within a virtual host. (See “Defining and deploying Web applications” on page 3-3 for a description of a Web application.) Web applications support the programming model concepts introduced to Web developers in the Servlet V2.1 and V2.2 API Specifications (see “Required JavaServer Pages levels” on page 1-3). These specifications expose a ServletContext object that provides a view of an application to each component at run time. The ServletContext object enables components within a Web application to locate, dispatch requests to, and share information with their related components at run time without knowing where they are physically located.

The components in a Web application:

- May be related in the sense that they work together to perform a business function. For example, a customer service Web application might contain servlets, JSPs, and HTML files that collaborate to provide this service to end users. Web applications are managed as a single unit.
- May be related only in the sense that you want to administer them as a single unit deployed on a particular Application Server and sharing a single ServletContext object. For example, if you are migrating from a previous version

of the Application Server, you might want to initially place all of your existing servlets and JSPs in a single Web application.

Configuring a virtual host

Virtual hosting allows a single Web server to act as a server for more than one internet host. The use of virtual hosting reduces the number of physical machines required, and reduces the need to dedicate and manage Web servers configured to non-standard ports. For example, a Web server listening on port 80, the usual port used for HTTP requests, may actually be servicing requests for hosts **www.mycompany.com** and **www.MyOtherCompany.com**. In this case, both host names would be registered in the Domain Name Server (DNS) with the same IP address. Information that the HTTP protocol requires to be in the input request enables the Web server to distinguish the intended virtual host.

The IBM HTTP Server for OS/390 contains support for logically partitioning its configuration based on the target host name of a request it is processing. The Application Server configuration file (was.conf) allows virtual hosts to be defined. Once defined, you can deploy one or more Web applications into a virtual host. This capability allows the Application Server configuration to be partitioned in accordance with the hosts for which it is servicing requests.

Properties of the form **host.**<*virtual-hostname*>**.**<>**=**<*value*> are used to define a virtual host. These properties indicate the name by which this host is known within the Application Server administrative domain (*virtual-hostname*).

A virtual host defined using this property may subsequently be used in a **deployedwebapp.**<*webapp_name*>**.host** property to deploy a Web Application under that virtual host.

The Application Server uses the **host.** properties to determine which virtual host to route an input HTTP request to. It checks the URL used to initiate an input request and routes the request to the specified virtual host.

The following properties are used to define a virtual host:

- **host.**<*virtual-hostname*>**.alias**<*hostname*>|localhost. This property is the name by which this host is known to clients and applications. The host may be specified as both a host name and target port.
- **host.**<*virtual-hostname*>**.mimetypefile**. This property is the fully qualified name of a file containing definitions for MIME types that describe the content that can be included in HTTP responses served from this host. The Application Server contains a default MIME type file containing standard MIME type definitions. The name of this default file is contained in the default was.conf file provided with the Application Server.

A host can have more than one alias. The alias definition may contain both a host name and a port number. When a client requests a Web application, servlet, or related resource, the Application Server compares the hostname and port in the request with the list of configured DNS aliases. If a match is not found, the servlet engine reports an error that is returned to the browser. The following illustrates how you might define the virtual host MyHost with DNS aliases of www.mycompany.com and www.MyOtherCompany.com: within a was.conf file:

```
host.MyHost.alias=www.mycompany.com
host.MyHost.alias=www.MyOtherCompany.com
```

See Appendix B, “was.conf file template”, on page B-1 for a complete description of the was.conf file properties that are applicable to defining a virtual host.

Note: The default_host provided with the product is meant to be used when running the Installation Verification Program (see “Invoking the Installation Verification Program” on page 2-4). It should be removed when you begin deploying other Web applications.

Defining and deploying Web applications

A Web application is a grouping of Web components, such as servlets, Java Server Pages (JSPs), and static files, such as HTML files and GIF files, that is defined and then deployed into a WebSphere virtual host and managed as a single unit. A virtual host is able to contain one or more Web applications.

Web applications enable you to manage components as a single unit even though they may not be related in a business sense. Components that do not explicitly use the ServletContext object and grouping concepts can still be connected and deployed as a single Web application. This is similar to how they were managed on previous versions of the Application Server; as a single unit deployed on a particular Application Server sharing a single ServletContext object.

There are two distinct sets of attributes associated with a Web application:

- Deployment attributes, which define the physical residency and characteristics of the Application Server into which the Web application is being deployed, such as the fully qualified directories that comprise the Web application’s classpath. These attributes are defined using was.conf file properties that start with the keyword **deployedwebapp** and are of the form:

  ```
  deployedwebapp.<webapp-name>.<property>=<value>
  ```

  <webapp-name> is the unique name by which a Web application is identified to the Application Server.

  The deployment attributes enable the ServletContext object to correctly resolve these physical residencies and characteristics at run time and efficiently deploy the Web application.
- Definitional attributes, which define the characteristics of a Web application, such as the servlets that are part of that application, and servlet and JSP mapping information. A web application’s definition can be communicated to the Application Server using either of the following methods:
  - Using the Web application was.conf file properties that start with the keyword **webapp** and are of the form:

    ```
    webapp.<webapp-name>.<property>=<value>
    ```

    <webapp-name> is the unique name by which a Web application is identified to the Application Server. (The same name is used on both the **deployedwebapp** and **webapp** properties associated with a specific Web application.)

    See Appendix B, “was.conf file template”, on page B-1 for a description of all of the **webapp** properties.
  - Using a Web application XML file, of the form *<webapp-name>***.webapp**, which uses XML tags to define the attributes of the Web application. Use a Web

Archive (WAR) file conversion utility to produce this Web application XML file. (See "Using Web applications contained in War files" on page 3-11 for more information about WAR files.)

For more information about the XML tags that can be used to provide definitional attributes for a Web application, see the Deployment Descriptor Elements section in the Java Servlet Specification V2.2 at URL:

```
http://java.sun.com
```

Whenever the Application Server initializes and detects Web application **deployedwebapp** properties in the was.conf file, it:

1. First looks for **webapp** properties in the was.conf file with the same value for the <*webapp-name*> keyword as appeared on the **deployedwebapp** properties. If the Application Server finds **webapp** properties that meet this requirement, it uses them to define the Web application and initialize the servlet. It will ignore any XML file of the form <*webapp-name*>**.webapp** with the same <*webapp-name*> keyword as appeared on the **deployedwebapp** properties.

   If **webapp** properties are being used to pass initalization parameters to a servlet, the servlet must include the statement:

   ```
   String s = getInitParameter("<parameter>")
   ```

   where <*parameter*> is the name of the parameter to be retrieved.
2. If the Application Server does not find **webapp** properties in the was.conf file with a matching value for the <*webapp-name*> keyword, it searches the Web application's classpath for a an XML file of the form <*webapp-name*>**.webapp**. If it finds such an XML file, the Application Server uses the contents of this file to configure the Web application's definition.

   If a .webapp XML file is being used to pass initalization parameters to a servlet, you must include the following property in the was.conf file:

   ```
   deployedwebapp.Payroll.classpath=<path>
   ```

   where <*path*> is the location of the Payroll application. (The associated .webapp file must be placed in the same classpath as the Web application.)

   For more information about defining a Web application in a .webapp file, see the Deployment Descriptor Elements section in the Java Server Specification V2.2 at URL:

   ```
   http://java.sun.com
   ```
3. If the Application Server does not find an appropriate .webapp file, it will look for a <*servlet-name*>.servlet file, where<*servlet-name*> is the name of the servlet requiring the initalization parameters. This .servlet file is an XML servlet configuration file and contains the name of the servlet class file, servlet initalization parameters, and a page list containing the URIs (universal resource identifiers) of the JSPs the servlet can call.

For more information about passing initalization parameters to a servlet, see "Passing init-parameters to a servlet" on page 3-26.

Note: If you are migrating from a previous version of the Application Server, you might want to initially define all of the Web components you are migrating within a single Web application. (See Appendix A, "Migrating from previous Versions of the Application Server", on page A-1 for more information on migrating existing Web components into the Web application structure.)

Including Web components in a Web application

A Web application can contain the following types of Web components:

- **Servlets**. The deployed Web application is able to host servlets that conform to the Java Servlet API 2.1 or 2.2 Specification. The Servlet API version is set for the Application Server; not for individual deployed Web applications. The value specified on the **appserver.compliance.mode** property in the was.conf file determines which version the servlets must conform to (see “Maintaining compatibility with existing applications” on page 2-9).

  Servlets that have been developed and tested on WebSphere Version 3.x for distributed platforms should be able to be deployed unchanged to this version of the Application Server except where an explicit restriction is noted. See Appendix D, “Programming Model Restrictions”, on page D-1 for a description of these restrictions.

  Servlet implementations can be provided to the Application Server as a java .class file, a java .ser file, or as part of a .jar file.

  Note: If you are intending to use servlets written to the Java Servlet API 2.01 Specification with this version of the Application Server, see Appendix A, “Migrating from previous Versions of the Application Server”, on page A-1 for more information on how to determine if changes are required before deploying them as part of a Web application.

- **Java Server Pages (JSPs) and JHTML files**. The deployed Web application is able to host Java Server Pages (JSPs) and JHTML files that comply with either the Java Server Pages 0.91, 1.0, or 1.1 Specifications. However, the Application Server does not support deploying JSPs written to different specification levels within the same Web application. JSPs written to different specification levels must be defined in separate Web applications. Use the **webapp.**<*webapp-name*>**.jsplevel=**<*JSP-spec-level*> property in the was.conf file to define the specification level of the JSPs contained in a specific Web application.

  The Application Server treats JSPs and JHTML files as text files for the purpose of interpreting and compiling them. At run time, the Application Server uses Java readers and writers to access the source files. The Java facilities perform the necessary conversion of the file content from the file encoding scheme (that is specified as a Java system property) to unicode format for processing. Therefore, JSP and JHTML files should be stored on the file system in the encoding scheme that is expected by the Java Virtual Machine (JVM).

  For JSP Specification levels 0.91 and 1.0, the JVM is typically configured with a file encoding of EBCDIC; for level 1.1, it is typically configured with a file encoding of ISO-8859–1 (ASCII). When transferring JSP and JHTML from an ASCII-based system, you must perform the necessary conversions as part of transferring the files.

  Notes:

  1. If you are using JDK 1.3, the Application Server’s JSP implementation enables you to leverage the newer 32K branch mechanism for JSPs that throw BranchTooLarge exceptions, by adding the **appserver.java.extraparm=-Djsp.largebranch=true** property to the was.conf file. This property causes the JVM system property, jsp.largebranch=true, to be passed to the JVM during JVM initialization.
  2. For a JSP written to the 0.91 Specification level, reloading automatically occurs when the Application Server detects that the JSP has been changed; for a JSP written to the 1.0 or 1.1 Specification level, automatic reloading will only occur if the reloading function has been enabled in the Web application. See “Class loading and optional reloading” on page 3-21 and “Deploying a

Web application to the Application Server" on page 3-7 for more information about the automatic reloading function.

3. According to an interpretation of the JSP 1.0 specification section 2.13.1 for the usebean tag, if the object is not found in a specified scope, and the class specified names a non-abstract class that defines a public no-args constructor, then that class is instantiated and the new object reference is associated with the scripting variable and with the specified name in the specified scope. This behavior can be enabled by adding the following property to the was.conf file:

```
appserver.extraparm=-Dcom.sun.jsp.useBeanConstructorMethod=true
```

- **Static Files**. The Application Server does not require static files, such as HTML and GIF files, to be in a Web application. They can remain on the hosting Web server. When the Application Server serves a static file, it treats it as a binary (ASCII) file and does not perform translation prior to writing it to the output stream. Therefore, static files, such as HTML and CSS files, that are not in binary format, should not be placed in a Web application. They should be stored on the file system in the codepage in which they are to be returned to the client.

To define the content and characteristics of a Web application, add properties of the form **webapp.**<*webapp-name*>**.**<*property*>**=**<*value*> to the was.conf file of the Application Server instance that is to host the Web application. (*webapp-name* is the logical internal name of the Web application being defined.)

These properties enable you to specify such things as a Web application's JSP specification level, URI mappings for included Web components, and definitions of the servlets that are to be included. They also enable you to define a custom error page for each Web application. Following are some of the properties that can be used to define a Web application. Appendix B, "was.conf file template", on page B-1 provides a complete description of all of the available properties.

webapp.<*webapp-name***>.jsplevel=<***JSP-spec-level***>**
This property is used to define the level of the JSP processor that is to be configured for this Web application.

webapp.<*webapp-name***>.attributes=errorrootcause=true | false**
Use this property to specify whether or not you want the error stack trace for this Web application to be returned to the browser from which it was requested. The default value for this property is false.

webapp.<*webapp-name***>.<***component-type-mapping***>=<***URI-pattern***>**
Use this property to provide the Application Server with information about the type of Web components that are represented by a particular URI mapping. Using this property, you might specify that URIs ending with .html or .gif should be treated as static files, or you might designate a pattern for Java Server Pages, JHTML files, or servlets. For *component-type-mapping* specify either **filemapping**, **servletmapping**, or **jspmapping**.

webapp.<*webapp-name***>.servlet.<***servlet-name***>.<***property***>=<***value***>**
Properties with this format are used to define a servlet that is included in the Web Application and include:

- **webapp.**<*webapp-name*>**.servlet**.<*servlet-name*>**.servletmapping=**<*URI-pattern*>, which is used to specify the servlet path relative to the Web application's root URI.
- **webapp.**<*webapp-name*>**.servlet**.<*servlet-name*>**.code=**<*servlet-class*>, which is used to associate a servlet with a class file.

- **webapp.**<*webapp-name*>**.servlet.**<*servlet-name*>**.autostart=**<*true/false*>, which is used to indicate whether or not a servlet is to be loaded whenever the Application Server starts.

Appendix B, “was.conf file template”, on page B-1 describes all of the properties that can be used to define a servlet within a Web application.

webapp.<*webapp-name***>.errorpagemapping=<***URI-pattern***>**
This property is used to define a custom error page for each Web application. A custom error page allows you to tailor the output provided to end users in the event of an application error. For example, you could add information indicating that Function A has experienced an error and to please call a provided number for help. The Application Server provides a default error reporting format that can be overwritten.

Note: If you do not include at least one of the following mapping properties in a Web application’s definition, the Application Server will not be able to initialize that application:
- **webapp.**<*webapp-name*>**.servletmapping=/servlet**
- **webapp.**<*webapp-name*>**.jspmapping=*.jsp**
- **webapp.**<*webapp-name*>.**filemapping=/**
- **webapp.**<*webapp-name*>**.servlet.**<*servlet-name*>**.servletmapping=/**<*servlet-name*>

Deploying a Web application to the Application Server

Before deploying Web applications on the Application Server, you must
1. Ensure that the Web components are in a format suitable for deployment.
2. Place the files containing the Web components on the Application Server.
3. Define the physical files to the Application Server by providing a Web application definition within the Application Server’s was.conf file.

One or more Web applications can be deployed into a virtual host on an Application Server. For each Web application, the following properties are used to deploy it in the Application Server. They all have the format **deployedwebapp.**<*webapp-name*.><*property*>**=**<*value*> and are added to the was.conf file of the hosting instance of the Application Server. *webapp-name* is the name of the Web application.

deployedwebapp.<*webapp-name***.>description=<***string***>**
This property is used to provide a description of a Web application, which can be exposed by invoking the Show server configuration example from a browser (see “Invoking the Installation Verification Program” on page 2-4).

deployedwebapp.<*webapp-name***>.host=<***virtual-hostname***>**
This property specifies the administrative name of the virtual host in which a Web application is deployed.

deployedwebapp.<*webapp-name***>.rooturi=<***value***>**
This property provides the pattern by which a Web application is known within its virtual host. For example, a Web application with a root URI of **/catalog** that is deployed within a virtual host with an alias of **www.mycompany.com** would be the target of requests for resources with the following URLs:

```
www.mycompany.com/catalog/Servlets/MyServlet
www.mycompany.com/catalog
www.mycompany.com/catalog/index.html
```

When multiple Web applications are defined with in a single virtual host, each must have a unique root URI pattern specification.

deployedwebapp.<*webapp-name*>.documentroot=<*directory-root*>
This property is used to specify the fully qualified name of the directory root in the file system that contains the files associated with the Web application, such as Java Server Pages, JHTML, and static files that are to be served by the Web application. The Application Server uses the directory specified on this property to locate a Web component of the specified type. Because the Application Server attempts to access JSPs, and JHTML and static files using the identity of the hosting Web server's process, the hosting Web server requires permission for read access to any files or directories which contain JSP, JHTML, or static files. For more information on how Web component requests are resolved to a physical resource, see "Mapping URLs to Web components" on page 3-17.

deployedwebapp.<*webapp-name*>.classpath=<*value*>
This property specifies the classpath that the application level class loader searches for a servlet when the system class loader cannot find it. This property is **REQUIRED** and must specify the directory where the servlet resides.

When the Application Server needs to load a servlet, it first attempts to load it using a system class loader, which is configured to search the classpath defined in the **appserver.classpath=**<your-libraries-for-the-jvm-classpath> property in the was.conf file. If the requested servlet cannot be found by the system class loader, an application-specific class loader is used to locate and load the target servlet implementation, using the classpath specified on the **deployedwebapp.**<*webapp-name*>**.classpath=**<*value*> property. (The Application Server internally maintains a separate class loader for each Web application instance.)

The Application Server will attempt to load servlets from the file system using the hosting Web server's identity. Therefore, the user-id under which the Web server runs requires permission for read access to any files or directories that contain servlet implementation files.

The Web application classpath specification can include directories, JAR files, or ZIP files. "Configuring Web applications" on page 3-14 describes where specific types of Web application files need to be placed. describes the proper placement of Web application files in more detail.

Notes:

1. The following types of classes must be added only to the **appserver.classpath=**<*your-libraries-for-the-jvm-classpath*> property:
   - Classes referenced from servlets whose objects are added to sessions. Such objects are serialized and their classes must not be reloaded. See "Class loading and optional reloading" on page 3-21 for more information on servlet reloading.
   - Classes that call Java Native Interface (JNI) methods. Those classes, and any imported classes, must be placed in the Application Server classpath to prevent loading errors.

   Note: Any text file placed in this classpath must be in EBCDIC format.
2. Classes that are added to the **appserver.classpath=**<your-libraries-for-the-jvm-classpath> property should not reference other classes that cannot be found in this classpath, such as classes included in the **deployedwebapp.**<*webapp-name*>**.classpath=**<*value*> property.

3. If your servlets use classes from Development Kit .jar files, such as JRIO classes in the recordio.jar file, you must make sure that these .jar files are included in the appserver.classpath rather than in the deployedwebapp.classpath. You can use the ″Show Server Configuration″ example in the Installation Verification Program to see which .jar files are currently in the appserver.classpath.

deployedwebapp.<*webapp-name*>.autoreloadinterval=<*millisecs*>
This property is used to specify whether or not a Web application is to be reloaded if changes are detected in the implementation file for one or more servlets in this Web application. The integer value <*millisecs*> indicates the number of milliseconds the Application Server waits between checks. The default value for <*millisecs*> is **0** and indicates that reloading will not be done. If you want to enable the auto reload function, carefully chose a reload interval that meets your requirements. System resources required to check for changes can be quite high. Therefore, the number of checks performed should be minimized.

This version of the Application Server reloads servlets on a deployed Web application basis. The Application Server is able to monitor servlets, loaded by the application specific class loader, for changes in the implementation file. If changes are detected in the implementation file for one or more servlets in a Web application for which reloading has been enabled, then all servlets and JSPs have been currently loaded by the application level class loader for that deployed Web application are reloaded. See “Class loading and optional reloading” on page 3-21 for more information about the servlet reloading.

Notes:

1. Servlet reloading is not recommended for production environments because of the impact it may have on system performance.
2. Versions 1.x of the Application Server, only support Version 0.91 JSPs and these JSPs are automatically reloaded even if the automatic reload function is not enabled. For Version 3.5, Version 0.91 JSPs are still automatically reloaded even if the automatic reload function is not enabled. However, Version 1.0 and Version 1.1 JSPs are only reloaded if automatic reloading is enabled for the Web applications in which they are included.

deployedwebapp.<*webapp-name*>.authresource.<*resource-name*>=<*servletmapping*>
This property is used to define security constraint policies for Web components. These policies allow the Application Server to perform access checks using operating system SAF facilities prior to accessing a Web component. <*resource-name*>is the name of the resource that is to be used along with the names of the virtual host and Web application to construct the SAF resource name of the form:

```
<virtual-hostname>.<webapp-name>.<resource-name>
```

<*servletmapping*> is the servlet mapping of the resource that is to be covered by the security constraint. (See “Securing Web components” on page 3-19 for more information on defining security policy.)

A detailed description of these properties is contained in the default was.conf file. See Appendix B, “was.conf file template”, on page B-1 for a copy of this file.

The following example illustrates the properties that might be added to the Application Server was.conf file to define and deploy the Web application, PayrollApp, into the virtual host, MyHost:

```
host.MyHost.alias=www.mycompany.com:8027
deployedwebapp.PayrollApp.host=MyHost
deployedwebapp.PayrollApp.rooturi=/Payroll
deployedwebapp.PayrollApp.classpath=/usr/MyCompanyApps/PayrollFiles/lib:
    /usr/MyCompanyApps/PayrollFiles/PayrollServlets.jar:/usr/MyCompanyApps
    /PayrollFiles/WEB-INF
deployedwebapp.PayrollApp.documentroot=/usr/MyCompanyApps/PayrollFiles
deployedwebapp.PayrollApp.autoreloadinterval=0
webapp.PayrollApp.filemapping=*.html
webapp.PayrollApp.filemapping=*.gif
webapp.PayrollApp.jspmapping=*.jsp
webapp.PayrollApp.jsplevel=1.0
webapp.PayrollApp.servlet.PayrollServlet.servletmapping=/Servlets/Payroll
webapp.PayrollApp.servlet.PayrollServlet.code=com.mycompany.payroll.
    MainServlet
webapp.PayrollApp.attributes=firstattr=Bill,secondattr=2
webapp.PayrollApp.servlet.PayrollServlet.initargs=x=0,y=Fred,z=true
```

Note: The **deployedwebapp.PayrollApp.classpath** property should be entered in the was.conf file as a single line. It was split here for printing purposes.

As this example illustrates, a clear distinction is made between the configuration data that needs to be supplied by the developer versus the configuration data that is supplied at deployment time. In this example, the **webapp.Payroll** definition properties the developer provided indicate that this Web application:

- Contains HTML, JSP, and GIF files.
- Includes JSPs that are coded to the JSP Version 1.0 Specification.
- Contains a servlet, PayrollServlet, whose implementation exists as MainServlet.class within the package com.mycompany.payroll.
- Includes mappings that indicate the URI pattern by which JSPs, GIFs and HTML files, and the PayrollServlet servlet are to be referenced at run time. (For example, the **webapp.PayrollApp.servlet.PayrollServlet.servletmapping** property specifies that **/Servlets/Payroll** is a valid URI pattern for the PayrollServlet servlet. Requests for **www.mycompany.com:8027/Payroll/Servlets/Payroll** will be serviced by the MainServlet implementation provided in the PayrollServlets.jar file.)

It also indicates that:

- All components within this Web application expect to be able to retrieve the values for firstattr and secondattr from their ServletContext object at run time.
- The servlet PayrollServlet expects three initialization arguments, x, y, and z to be available at run time.

After receiving the physical files and Web application definition (**webapp.Payroll** properties) from the developer, the deployer adds **deployedwebapp** properties to the was.conf file to define the Web application to the Application Server. These properties:

- Provide the classpath that, in conjunction with classpath specifications provided as part of other Application Server properties, is to be used to locate a servlet implementation class at run time.
- Indicate that auto-reloading of components within the Web application is not enabled (the **deployedwebapp.PayrollApp.autoreloadinterval** property has a value of 0).

- Indicate in which host(s) the Web application may be deployed.
- Provide the root URI by which the Web application is exposed to end users of the Web site.
- Define, to the Application Server, the physical location of the various Web components. (In this example, the deployer indicates to the Application Server that these files and JSPs should be resolved using the directory **/usr/MyCompanyApps/PayrollFiles** as the document root.
- Indicates that the class file for the servlet PayrollServlet has been packaged within the jar file, PayrollServlets.jar.

As previously described in this section, instead of using **webapp** properties, a developer can provide the development specific configuration data for a Web application definition in a separate XML document. See the following product file for examples of Web application definitions that are coded this way:

```
applicationserver_root/AppServer/hosts/default_host/examples/servlets/
     examples.webapp
```

Using Web applications contained in War files

The servlets and JavaServer Pages (JSP) files in a Web application share a servlet context, meaning they share data and information about the execution environment, including a Web application classpath. Version 3.5 of the Application Server introduces a new way to introduce Web applications into the WebSphere environment. The product now consumes and converts Web Archive (WAR) files into WebSphere configurations. Alternatively, you can continue to configure Web applications directly using properties in the Application Server was.conf file. The latter allows you to add WebSphere servlets to your Web applications to extend their functionality.

Web Archive (WAR) files are essentially JAR files containing the various files and configuration information of a Web application. (The WAR file is included as part of the Java Servlet specification.) WAR files are useful for importing complete Web applications into an Application Server run-time environment. To maintain compatibility with existing applications, WAR files are only used as a deployment vehicle. After a WAR file is installed into the Application Server run-time environment, the WAR file itself is no longer used.

While the Application Server does not support the direct importation of Web applications from WAR files, you can enter the following commands from the OMVS shell to set your JAVA environment variables, if they have not already been set:

```
export JAVA_HOME=/usr/lpp/java13/IBM/J1.3
export PATH=$PATH:/usr/lpp/java13/IBM/J1.3/bin
```

and then enter the following command to convert the WAR file into a .webapp file format that can be imported into a stand-alone Web server environment:

```
applicationserver_root/AppServer/bin/wartowebapp.sh   warfile-name
```

Note: **DO NOT** run the wartowebapp.sh program against .war files containing JSP 0.91 or 1.0 files.

The converted WAR file can then be deployed into the Application Server by merging the generated deployment properties into *applicationserver_root*/AppServer/properties/was.conf.

For example, if you enter the following command:

```
 $WAS_HOME/bin/wartowebapp.sh /u/user25/warfiles/jsp-tests.war
```

you will receive the following message:

```
/u/user25/WebSphere/AppServer/bin/wartowebapp.sh:
please enter value(s) for the following missing parameter(s):
```

and will be prompted for the name of the hosting virtual host, the name of the Web application contained in the WAR file, the file encoding of the local OS/390 machine, where you want to have the Web application installed, and :

```
VIRTUAL_HOST_NAME    <null to accept: default_host>
WEBAPP_NAME          <null to accept: jsp-tests>
WEBAPP_DESTINATION   <null to accept: $was_install_root$/AppServer/hosts/
    default_host>
WEBAPP_PATH          <null to accept: /webapp/jsp-tests>
WAS_HOME             <null to accept: /usr/lpp/WebSphere/AppServer>
LOCAL_FILE_ENCODING  <null to accept: en_US.IBM-104>
ENTER YES TO CONFIGURE THE AUTO-REGISTRATION (INVOKER) SERVLET
         <Code>com.ibm.servlet.engine.webapp.InvokerServlet</code>
         <servlet-path>/servlet</servlet-path>
          or enter no,
          or enter a servlet path (i.e. /servlet/* for compliance mode)
INVOKER_SERVLET       <null to accept: no>
```

If you accept the default values for all of these prompts, you will receive the following messages:

```
parameters in effect:

  TEMP_DIRECTORY              /tmp
  WAR_FILENAME                /u/user25/warfiles/jsp-tests.war
  VIRTUAL_HOST_NAME           default_host
  WEBAPP_NAME                 jsp-tests
  WEBAPP_DESTINATION          $was_install_root$/AppServer/hosts/default_host
  WEBAPP_AUTO_RELOAD_INTERVAL 0
  WEBAPP_PATH                 /webapp/jsp-tests
  WAS_HOME                    /u/user25/WebSphere/AppServer
  LOCAL_FILE_ENCODING         en_US.IBM-1047
java  -Xmx64m -classpath /u/user25/WebSphere/AppServer/lib/antxalan_1_1.jar:
/u/user25/WebSphere/AppServer/lib/xerces.jar:/u/user25/WebSphere/AppServer
    /lib/ant.jar:
/u/user25/WebSphere/AppServer/lib/ibmant.jar -Dwebapp.dest=
/u/user25/WebSphere/AppServer/hosts/default_host -Dwar.filename=
/u/user25/warfiles/jsp-tests.war -Dvirtual.host.name=
default_host -Dwas.home=/u/user25/WebSphere/AppServer -Dtmp.directory=
/tmp -Droot.uri=/webapp/jsp-tests -Dwebapp.name=jsp-tests -Dant.home=
/u/user25/WebSphere/AppServer -Dwas.java.name=
java  org.apache.tools.ant.Main -buildfile
/u/user25/WebSphere/AppServer/properties/convertwar.xml transform.to.webapp
Buildfile: /u/user25/WebSphere/AppServer/properties/convertwar.xml

init:

unpack.war:
    [unzip] Expanding: /u/user25/warfiles/jsp-tests.war into
        /tmp/jsp-tests

copyto.webapp:
    [mkdir] Created dir: /u/user25/WebSphere/AppServer/hosts/
        default_host/jsp-tests
    [mkdir] Created dir: /u/user25/WebSphere/AppServer/hosts/
        default_host/jsp-tests/web
    [mkdir] Created dir: /u/user25/WebSphere/AppServer/hosts/
        default_host/jsp-tests
        /web/WEB-INF
    [mkdir] Created dir: /u/user25/WebSphere/AppServer/hosts/
        default_host/jsp-tests/servlets
  [copydir] Copying 272 files to /u/user25/WebSphere/AppServer/
        hosts/default_host/jsp-tests/web
  [copydir] Copying 2 files to /u/user25/WebSphere/AppServer/
        hosts/default_host/jsp-tests/web/WEB-INF

copy.war.classes:
  [copydir] Copying 96 files to /u/user25/WebSphere/AppServer/
        hosts/default_host/jsp-tests/servlets

copy.war.lib:
  [copydir] Copying 1 files to /u/user25/WebSphere/AppServer/
        hosts/default_host/jsp-tests/servlets

transform.to.webapp:
  [deltree] Deleting: /tmp/jsp-tests

BUILD SUCCESSFUL

Total time: 11 seconds

iconv -f ISO8859-1 -t IBM-1047 /u/user25/WebSphere/AppServer/hosts/default_host/
   jsp-tests/servlets/jsp-tests.webapp > /tmp/wartowebapp-convert-work

created '/u/user25/WebSphere/AppServer/hosts/default_host/jsp-tests/
   was.conf.updates' containing:

deployedwebapp.jsp-tests.host=default_host
deployedwebapp.jsp-tests.rooturi=/webapp/jsp-tests
deployedwebapp.jsp-tests.classpath=$was_install_root$/AppServer/hosts/
   default_host/jsp-tests/servlets
deployedwebapp.jsp-tests.documentroot=$was_install_root$/AppServer/hosts/
   default_host/jsp-tests/web
deployedwebapp.jsp-tests.autoreloadinterval=0

please merge the contents of '/u/user25/WebSphere/AppServer/hosts/default_host/
   jsp-tests/was.conf.updates'
into '/u/user25/WebSphere/AppServer/properties/was.conf' to deploy the web
   application: jsp-tests
```

As the final message indicates, you must take the resulting **deployedwebapp** properties (highlighted in the previous example) and add them to the was.conf file for the Application Server on which you want this Web application to be deployed before attempting to use this Web application.

Note: With the exception of the .webapp file and .servlet files, XML content in the Web Application generated by $server_root$/AppServer/bin/wartowebapp.sh is **NOT** converted to the file encoding of the local OS/390 machine. This is likely to be a problem since most XML content found in a .war file is probably encoded using codepage ISO8859-1 and the WebSphere Application Server Version 3.5 expects XML content to be encoded in the native file encoding of the OS/390 machine. The following three step process can be used to convert all XML content from codepage ISO8859-1 to codepage IBM-1047. From the Unix System Services command line, enter the following 3 commands:

```
echo "iconv -f ISO8859-1 -t IBM-1047 \$* > /tmp/convert-work;
        cat /tmp/convert-work > \$*"  > /tmp/convert-script
chmod 755 /tmp/convert-script
find /u/joe/webapps/MyWebApp -name '*.xml' -exec /tmp/convert-script {} \;
```

/u/joe/webapps/MyWebApp is the location in the Unix System Services file system where the Web application was created by the

$server_root$/AppServer/bin/wartowebapp.sh tool. Each of these commands must be entered as a single line. The echo command is split here for printing purposes.

Configuring Web applications

When configuring Web applications, you should understand a few main settings:

- The classpath specifies where to find the servlets that belong to an application. The classpath can specify a directory containing servlets, or can specify each servlet explicitly. It can also specify the location of other files supporting the Web application.
- The document root specifies where to find the static content and JSP files belonging to the Web application.

You can also include **webapp** properties to specify:

- Servlet filtering parameters
- Affiliation with a virtual host
- Whether to reload servlets whose class files have changed

After Web applications are developed, add them and their building blocks (servlets, JSP files, and such) to the Application Server environment by preparing the WebSphere environment to handle them. This includes:

- Setting the Application Server (JVM) and Web application classpaths
- Placing property files in appropriate directories
- Placing application files in appropriate directories
- Configuring application settings
- Securing applications (see “Securing Web components” on page 3-19)

Setting the Application Server (JVM) and Web application classpaths

The Application Server environment has two classpath components:

1. The Application Server (JVM) Classpath
2. The Web application Classpath

The following table describes the JVM classpath.

Where to set	There is one Application Server classpath for each application server in the Application Server environment. Each Application Server corresponds to a JVM. Set the classpath using the Application Server run-time properties contained in the Application Server was.conf file. (See “Specifying configuration properties” on page 2-6, and Appendix B, “was.conf file template”, on page B-1 for a description of these properties.)
Scope	This classpath is visible to all servlets and JSP files contained by an Application Server.
Behavior	The classes in this classpath are loaded by the JVM ClassLoader. Therefore, after the Application Server is started, any changes to this classpath will not take effect until the Application Server is stopped and started again.
Reloadable?	Classes loaded from this classpath will not be reloaded if they are changed while the Application Server is running.

Typical contents	• Classes referenced from servlets whose objects are added to sessions. Such objects are serialized and their classes must not be reloaded. • Classes that call Java Native Interface (JNI) methods. These classes and any imported classes must be placed in the Application Server classpath to prevent loading errors. • Classes common to all Web applications, such as JOBL driver classes. • Files that must remain in EBCDIC format. (It is recommended that files be converted to ASCII format, if possible, and placed in the Web application classpath specified on the **deployedwebapp.**<*webapp-name*>**.classpath=** property contained in the Application Server was.conf file.) All files contained in this classpath must be in EBCDIC format. Classes that are put in this classpath should not reference other classes that cannot be found in this classpath.

The following table describes a Web application classpath.

Where to set	Use the **deployedwebapp.**<*webapp-name*>**.classpath=** property contained in the Application Server was.conf file. (See “Specifying configuration properties” on page 2-6, and Appendix B, “was.conf file template”, on page B-1 for a description of this property.)
Scope	The classpath is visible to all servlets and JSP files in the corresponding Web application.
Behavior	If reloading is enabled, this classpath is monitored and all components (JAR or class files) are reloaded whenever it is detected that a servlet has been updated. A new JAR file is automatically loaded upon detection in any directory already contained in this classpath. This means that it is not necessary to explicitly specify a new JAR file in this classpath. It is sufficient to just put the JAR file in a directory that is already present in this classpath. Remote servlet loading (that is, loading servlets across a network) is not supported. All application components must be on the machine containing the Application Server hosting the application.
Reloadable?	Yes
Typical contents	• Directories or JAR files with servlet classes • Directories or JAR files with helper classes that are not included in the servlet JAR file and that are expected to be reloadable; • Directories or JAR files with access bean classes that are referenced from servlet classes. All files contained in this classpath must be in ASCII format. Files that are in EBCDIC format must either be converted to ASCII format or placed in the Application Server (JVM) classpath.

The following table summarizes the directory path that should be specified for the various types of files.

File description	**File Extension**	**Directory path**
HTML documents and related static files	.html, .jhtml, .gif, .au, and so on	These can be either served by the Web server, or placed in the Web application document root with the WebSphere file servlet enabled.
JavaServer Pages files	.jsp	Web application document root.
Servlet that are not to be reloaded	.class or .jar	Application Server classpath. If the servlets are in a package, mirror the package structure as subdirectories under the Web application classpath.
Servlet configuration file	.servlet	Web application classpath.

File description	File Extension	Directory path
JavaBean (not an enterprise bean) or other object to be reloaded	.ser or .jar	Web application classpath
JavaBean (not an enterprise bean) or other object **not** to be reloaded, such as serialized objects and servlets that use Java Native Interface methods	.ser or .jar	Application Server classpath
Java objects added to a session	.class, .jar, or .ser	Application Server classpath. This requirement applies to non-EJB objects in either of the following conditions: • Session persistence is enabled (the default setting). • The Application Server is part of a session cluster. In a session cluster, be sure to place the objects in the application server classpath on each cluster host and cluster client. An object in the Application Server classpath is not reloaded when its source file changes.
Objects passed as arguments for remote calls		Application Server classpath

Placing .property files in appropriate directories

All .properties files accessed by the Application Server, must reside in one of the following directories:

- The *applicationserver_root*/**AppServer/properties** directory.
- The *applicationserver_root*/**AppServer/lib** directory.
- A directory specified on a **deployedwebapp.**<*webapp-name*>**.classpath** property in the was.conf file.

Where you place your .properties files depends on how they will be loaded:

- If a .properties file will be loaded via the **appserver.classpath** property in the was.conf file, it must be included in either the *applicationserver_root*/**AppServer/lib** or *applicationserver_root*/**AppServer/properties** directory. All .properties files that are loaded this way must be in EBCDIC format.
- If a .properties file will be loaded via a **deployedwebapp.**<*webapp-name* property, it's location must be specified on a corresponding **deployedwebapp.**<*webapp-name*>**.classpath** property. All .properties files that are loaded this way must be in ASCII (ISO8859-1) format.

When looking for a .properties file, the Application Server first searches the classpath specified on the **deployedwebapp.**<*webapp-name*>**.classpath** property, and then searches the classpath specified on the **appserver.classpath** property.

Placing application files in appropriate directories

For successful deployment of Web applications, you must specify where the Application Server can find the files belonging to these applications. The following table indicates where to specify specific types of content.

Type of content	For example...	Where to specify
Java content	JAR and class files for servlets	Application server classpath or Web application classpath
Static content	JSP files, HTML files, graphics, and so on	Web application document root

For successful operation of applications, you must place the application files in the appropriate product directories, and set classpaths and document roots as necessary, before starting the Application Server that will be hosting that application.

Mapping URLs to Web components

Resources are defined to the Application Server by a Uniform Resource Locator (URL). URLs provide Web paths for identifying the location of servlets, Web pages, JSP files JSP files and other Web components such as image files.

To resolve a URL to an individual Web component, the Application Server incrementally parses pieces of the URL. A URL can be of the form:

```
<protocol>//<DNS-hostname><RequestURI>
```

- *<DNS-hostname>* is the name of the virtual host hosting the requested Web application. The Application Server uses this part of the URL to route the request to the proper virtual host.
- *<RequestURI>*provides path information and can be of either of the following forms:

  ```
  <ContextPath><PathInfo>
  <ContextPath><ServletPath><PathInfo>
  ```

The *<ContextPath>* part of the URL is used to route the request to the selected Web application deployed under the given virtual host. Depending on the Web application definition, *<RequestURI>*s that include a *<ServletPath>* are assumed to be requests to run servlets. The *<PathInfo>* part of the path is used to identify the requested Web component within the deployed Web application, and will be resolved to a physical entity.

For a servlet request the following methods in the HttpServletRequest interface can be used to access the parts of the *<RequestURI>*:
- getContextPath
- getServletPath
- getPathInfo

For example, the following URL might be used to request a servlet:

```
http://MyCompanyHost/PayrollApp/servlet/PayrollServlet
```

For this request,
- **PayrollServlet** is the *<PathInfo>* for a servlet defined in a Web application.
- **/servlet** is the *<ServletPath>*which is used by the Web application to indicate that <RequestURI>s that start with **/servlet** are assumed to be a request for a servlet.
- **PayrollApp** is the *<ContextPath>* configured for the deployed Web application
- **MyCompanyHost** is the *<DNS-hostname>*

Understanding how the Application Server resolves a resource request entered by a client is key to developing a strategy for managing your Web components, and the name space by which they are exposed to other applications and the outside world. Resources are defined to the Application Server by a Uniform Resource Locator (URL). URLs themselves consist of a virtual host name concatenated with a Uniform Resource Identifier (URI). The URI is a concatenation of the Web application name and the Web component name. To resolve a URL to an individual Web component, the Application Server incrementally parses pieces of the URL. Specifically, the Application Server:

1. Determines the virtual host that should process the request.
2. Determines the target Web application within that host.
3. Determines the requested Web component and resolves it to a physical entity.

Determining the virtual host that should process a request

The Application Server assumes that the left-most portion of the URL, up to and including the character that precedes the first slash, is the host mapping. The Application Server then attempts to map this host mapping to a host definition statement contained on a **host.<virtual-hostname>** property in the Application Server's was.conf file. For example, the following properties might be used to define the host, **MyCompanyHost**, so that it matches host mappings of **www.mycompany.com** and **www.mycompany.com:8027**:

```
host.MyCompanyHost.alias=www.mycompany.com
host.MyCompanyHost.alias=www.mycompany.com:8027
```

If the Application Server finds a matching host name, processing continues to the next step. If a matching host name is not found, the request is rejected and an error code is issued indicating that the resource was not found.

Determining the requested Web component and resolving it to a physical entity

The final processing the Application Server performs during the URL mapping process is dependent upon the type of Web component requested:

- **If the target Web component is a JSP or JHTML file** - the Application Server attempts to locate the matching source file within the Application Server document root structure. If the JSP is located, the Application Server processes the JSP and compiles it into a servlet. If a corresponding JSP source file is not found, an error code indicating that the resource was not found is returned to the client. If an error is encountered while compiling the resultant servlet, a response code of 500 is returned to the client. If a compilation error occurs, appropriate error information is written to the ncf.log file.
- **If the target Web component is a static file** - the Application Server attempts to locate the corresponding file within the document root structure of the Application Server. If the file is successfully located, the Application Server writes the content of the file to the output stream. (The Application Server considers the file content to be binary data. Therefore, translation is not performed on the content prior to writing it to the output stream.) If the corresponding physical file is not found, a response code indicating that the resource is not found is returned to the error log for processing. When the Application Server is able to locate the corresponding physical file but encounters a problem when it tries to access it (i.e., insufficient access authority), a response code of 500 is returned and the appropriate error information is written to the ncf.log file.
- **If the target Web component is a servlet** - the Application Server attempts to resolve the resultant servlet name to a file containing the servlet implementation by trying to locate a .servlet file containing additional configuration information

for this servlet. .servlet files are XML documents that may be placed in the classpath. If a .servlet file is located, the Application Server uses the information within it to locate, load, and initialize the target servlet. The following example shows the grammar that exists within a .servlet XML document file. It illustrates a servlet implementation that exists as class **MainServlet** within the package **com.mycompany.payroll**. The file also contains the parameter information that is to be provided to the servlet as part of its servlet configuration.

```
<?xml version="1.0"?>
        <servlet>
           <code>com.mycompany.payroll.MainServlet</code>
              <init-parameter value="5" name="x"/>
              <init-parameter value="abc" name="y"/>
         </servlet>
```

The WebSphere Studio product makes use of .servlet files when publishing servlets to the Application Server. If a .servlet file is not located, the Application Server attempts to locate a servlet implementation with a class name equal to the servlet name. If a servlet implementation is unable to be located, a response code indicating that the resource was not found is returned to the error log for processing.

Securing Web components

The Application Server allows Web resources to be uniquely identified such that access control policies can be applied to them. This capability, along with the user authentication capabilities of the hosting Web server, provide a mechanism for controlling access to Web resources at various levels of granularity on behalf of users and/or groups of users. The Application Server does not provide support for authenticating users. Instead, it relies on the hosting Web server to perform this function. See your Web server documentation for more information on how to implement security.

The Application Server provides the ability to define security constraints within a deployed Web application. A security constraint allows specific servlet mappings within a Web application to be associated with a resource name for use in the administrative domain. The following example illustrates a security constraint which associates the administrative name **MyServletResources** with requests that map to pattern **/Servlets** within the specified Web application:

```
host.MyCompanyHost.alias=www.mycompany.com:8027
deployedwebapp.PayrollApp.host=MyCompanyHost
deployedwebapp.PayrollApp.description=Payroll Application for My Company
deployedwebapp.PayrollApp.rooturi=/Payroll
deployedwebapp.PayrollApp.classpath=/u/MyCompanyApps/PayrollFiles
        /lib:/u/MyCompanyApps/PayrollFiles/classes
deployedwebapp.PayrollApp.documentroot=/u/MyCompanyApps/PayrollFiles
webapp.PayrollApp.servletmapping=/Servlets
webapp.PayrollApp.jsplevel=1.0
webapp.PayrollApp.jspmapping=*.jsp
webapp.PayrollApp.filemapping=*.html
webapp.PayrollApp.filemapping=*.gif
deployedwebapp.PayrollApp.authresource.MyServletResources=/Servlets
```

Notes:

1. For this example, the Application Server is running in compatibility mode (see “Maintaining compatibility with existing applications” on page 2-9). If the Application Server was running in compliance mode, the **servletmapping** property would specify **/Servlets/*** instead of **/Servlets**.

2. The values specified on the **deployedwebapp.PayrollApp.classpath** property must all be in a single line in the was.conf file. It was split here for printing purposes.

As part of request processing, the Application Server attempts to determine if there is a security constraint that matches the requested URL. When a match is detected, the Application Server performs an authorization check using SAF facilities prior to accessing the resource. (An authorization check is not performed for requests which do not have a matching security constraint.) If the authorization check is successful, request processing continues. If the access control check fails, HTTP response code 500 is returned to the client indicating that access to the resource is forbidden.

Authorization checks are performed against resource profiles defined within the SOMDOBJS RACF class. It is your responsibility to define the appropriate resource profiles within this RACF class. The resource profile name is constructed from the combination of the administrative names of the virtual host, the deployed Web application, and the resource name of the matching security constraint. The components are separated by a period. For the purpose of authorization checking, the preceding example would have resulted in requests for **www.mycompany.com:8027/Payroll/Servlets/MyServlet1** resolving to the resource **MyCompanyHost.PayrollApp.MyServletResources**.

The Application Server does not verify whether the resultant resource profile conforms to SAF conventions. It is your responsibility to ensure that the administrative names consist of characters that result in a valid SAF resource definition. However, before performing an authorization check, the Application Server does convert all alphabetic characters contained in a resource profile to uppercase.

The level of access check is determined by the type of HTTP request that is being processed. HTTP PUT requests result in checking whether the current principal has UPDATE access to the specified resource; HTTP DELETE requests result in checking whether the current principal has ALTER access to the specified resource. All other HTTP methods result in a check for READ access to the specified resource.

When the Application Server is required to access a physical resource as part of request processing, it performs the access using process level identity. The access is performed from an execution thread which has an identity that reflects the identity of the hosting Web server process for that Application Server instance. Therefore, you must ensure that the Web server process has been granted access to the necessary files at run time. In particular, Web server processes must have read access to files within the document root, as well as any Java class and jar files that may need to be loaded.

If, while accessing a physical resource, the Application Server fails to access a physical file, an HTTP Response code 500, indicating an internal server failure has occurred, will be returned to the client.

Class loading and optional reloading

Servlets and non-servlet Java components, such as JSPs, of an application are loaded by the Application Server's Web application classloader or the system classloader (the JVM classloader). The following points explain the timing of the loading, which classloader loads the Java component, and whether the component will be reloaded when a change is detected:

- The Application Server classpath is dynamically set when you start the product. The default setting for the classpath contains all of the Application Server APIs (the JAR files in the *applicationserver_root***AppServer/lib** directory). When the Application Server starts, the system classloader automatically loads the classes in the Application Server classpath. Those classes are not reloadable.
- You must add the following Java components to the Application Server classpath:
  – Java objects that are added to sessions. Such objects are serialized and must not be reloaded.
  – Java classes that call Java Native Interface (JNI) methods. Those classes and any imported classes must be placed in the Application Server classpath to prevent loading errors.
  – Objects passed as arguments for remote calls.
- When you configure a Web application, you specify the application classpath, which contains the servlets and their non-servlet Java components. You can also use the following property to indicate whether or not you want the automatic reload function to be enabled for this Web application:

  ```
  deployedwebapp.<webapp-name>.autoreloadinterval=<millisecs>
  ```

  See "Deploying a Web application to the Application Server" on page 3-7 and Appendix B, "was.conf file template", on page B-1 for more information about specifying this property.
- If automatic reloading is enabled (a non-zero value is specified for **autoreloadinterval**), the classloader monitors the application classpath and reloads all of the Java components in that application classpath whenever it detects that a loaded servlet has been updated.

Automatic reloading at the application level keeps all of the application components synchronized, and conserves system resources. If your application uses Java components whose classes you do not want to be reloaded, add those classes to the Application Server classpath instead of the application classpath. Then the classes will not be reloaded, but the objects will be.

When automatic reloading is in effect, all dispatched work (servlets, jsps, beans) that have reached a certain point in their processing will complete before a reload is initiated. Therefore, a reload must wait until this work finishes. All dispatched work that has not reached this point is halted, and must wait for the current work and the reload to complete. At that point processing of the dispatched work that had to wait resumes. However, all session objects associated with that work that had to wait for the reload to complete will be destroyed.

A Web application's scope is its application classpath plus the Application Server system classloader classpath. The Application Server does not support remote servlet loading (that is, loading servlets across a network). All of a Web application's components must be on the Application Server machine.

Note: If you are collecting session data while the reload function is enabled, whenever a servlet contained in a Web application is reloaded, all sessions associated with that application will be destroyed. However, the reloading of JSPs associated with that application will not cause these sessions to be destroyed.

Compiling JSP level 1.0 or level 1.1 source files

JSP source files coded to the level 1.0 or 1.1 specification level may be:

1. Compiled during the Application Server initialization process, and then loaded during Application Server JSP 1.1 processor initialization, or
2. Pre-compiled outside the Application Server initialization process. If JSP source files are pre-complied, the servlet mapping, which will be processed by the JSP level 1.0 or level 1.1support servlet, must be specified by:
   - Adding the appropriate <code></code> and <autostart></autostart> XML elements to the .webapp XML file associated with the Web application, or
   - Adding the appropriate webapp.xxx.servlet.yyy.code and webapp.xxx.servlet.yyy.autostart=true properties to the was.conf file, and
   - Ensuring that the JSP 1.0 or 1.1 support servlet is autostarted.

When the Application Server encounters these XML elements or was.conf properties, it checks to see if a generated .class file for the JSP already exists in the working directory. If one does not exist, it will generate Java code from the JSP source file, compile that code, and load the generated class file during JSP level 1.0 or 1.1 processor initialization.

JSPs can also be pre-compiled outside of the Application Server runtime.

Pre-compiling JSPs

Level 1.0 JSPs can be pre-compiled by running the jsp10BatchCompile.sh shell script; level 1.1 JSPs can be pre-compiled by running the jsp11BatchCompile.sh shell script. Both of these shell scripts are provided with the Application Server in the directory:

```
applicationserver_root/AppServer/bin
```

To invoke the jsp10BatchCompile.sh script, enter the following command (on one line) from the USS command line:

```
|   applicationserver_root/AppServer/bin/jsp10BatchCompile.sh
|           target-directory   full-path-to-jsp-file(s)
|           webapp-classpath   webapp-document-root
|   [-keepgenerated]
|   [-recurse]
```

where:

target-directory
is the directory into which the generated Java file will be compiled.

full-path-to-jsp-file
is the fully qualified path to the JSP source file.

webapp-classpath
is the Web application's class path.

webapp-document-root
is the Web application's document root.

keepgenerated
is optional and requests that the generated Java code be kept.

recurse
is usually optional and requests that all file(s) with extensions .jsp, .jsv, and .jsw contained in all subdirectories of full-path-to-jsp-file(s) be compiled. This parameter is **required** if full-path-to-jsp-file(s) includes a directory.

Note: The parameters webapp-classpath and webapp-document-root must match exactly the values specified on the **deployedwebapp.<webapp-name>.classpath** and **deployedwebapp.<webapp-name>.documentroot** properties in the was.conf file for the Web application requiring the JSP.

To invoke the jsp11BatchCompile.sh shell script, enter the following command (on one line) from the USS command line:

```
applicationserver_root/AppServer/bin/jsp11BatchCompile.sh
    target-directory   full-path-to-jsp-file
[uriRoot]
[-recurse]
```

where:

target-directory
is the directory into which the generated java file will be compiled.

full-path-to-jsp-file
is the fully qualified path to the JSP source file.

uriRoot
is the Web application's document root. This is parameter is optional if a directory named WEB-INF exists immediately underneath the document root. It is **required** if such a directory does not exist.

recurse
is usually optional and requests that all file(s) with extensions .jsp, .jsv, and .jsw contained in all subdirectories of full-path-to-jsp-file(s) be compiled. This parameter is **required** if full-path-to-jsp-file(s) includes a directory.

Both of these shell scripts call the org.apache.jasper.JspC (batch compiler) code that is shipped with the Application Server. This code checks for a .class file for this JSP and, if one does not already exist in the target directory, generates Java code from the JSP source and compiles that code. This batch compiler will not re-compile a JSP if a .class file for that JSP already exists and is not outdated.

The target-directory parameter must be consistent with the values specified in the was.conf file. It must be of the form xxx/yyy/zzz, where xxx is the value specified on the appserver.workingdirectory property, yyy is the value specified on the deployedwebapp.zzz.host property, and zzz is the name of the deployed Web application.

For example, if, for Web application pq61925, you included the following properties in the was.conf file:

```
appserver.workingdirectory=/tmp
deployedwebapp.pq61925.host=default_host
deployedwebapp.pq61925.rooturi=/webapp/pq61925
deployedwebapp.pq61925.classpath=
    /u/webusr1/sys02/webapps/pq61925/servlets
deployedwebapp.pq61925.documentroot=
   /u/webusr1/sys02/webapps/pq61925/web
deployedwebapp.pq61925.autoreloadinterval=1000
```

```
webapp.pq61925.jspmapping=*.jsp
webapp.pq61925.jspmapping=*.jsv
webapp.pq61925.jspmapping=*.jsw
webapp.pq61925.jsplevel=1.1
webapp.pq61925.filemapping=/
webapp.pq61925.servlet.ajsp.name=ajsp
webapp.pq61925.servlet.ajsp.description=This is ajsp
webapp.pq61925.servlet.ajsp.code=/a.jsp
webapp.pq61925.servlet.ajsp.servletmapping=/a.jsp
webapp.pq61925.servlet.ajsp.autostart=true
webapp.pq61925.servlet.bjsp.name=bsp
webapp.pq61925.servlet.bjsp.description=This is bjsp
webapp.pq61925.servlet.bjsp.code=/b.jsp
webapp.pq61925.servlet.bjsp.servletmapping=/b/b.jsp
webapp.pq61925.servlet.bjsp.autostart=true
```

or the .webapp file for this Web application contains the following information:

```
<?xml version="1.0"?>
<webapp>
   <name>pq61925</name>
   <servlet>
      <name>ajsp</name>
      <code>a.jsp</code>
      <autostart>true</autostart>
      <servlet-path>/a.jsp</servlet-path>
   </servlet>
<servlet>
      <name>bjsp</name>
      <code>/b/b.jsp</code>
      <autostart>true</autostart>
      <servlet-path>/b/b.jsp</servlet-path>
   </servlet>
   <servlet>
      <name>jsp11</name>
      <description>JSP 1.1 support servlet</description>
      <code>org.apache.jasper.runtime.JspServlet</code>
      <servlet-path>*.jsp</servlet-path>
      <servlet-path>*.jsv</servlet-path>
      <servlet-path>*.jsw</servlet-path>
      <autostart>true</autostart>
   </servlet>
</webapp>
```

If you issue the following command (all on one line):

```
/usr/lpp/WebSphere/AppServer/bin/jsp11BatchCompile
   /tmp/default_host/pq61925  /u/webusr1/sys02/web/a.jsp
```

The command line compiler JspC will check the target-directory, **/tmp/default_host/pq61925**, for a .class file for JSP a.jsp. If one does not exist, or if one exists but is out of date, it will compile the JSP a.jsp contained in directory **/u/webusr1/sys02/web**.

Similarly, if you issue the command (all on one line):

```
/usr/lpp/WebSphere/AppServer/bin/jsp11BatchCompile
        /tmp/default_host/pq61925  /u/webusr1/sys02/web/b/b.jsp
```

The command line compiler JspC will check the target-directory, **/tmp/default_host/pq61925/b**, for a .class file for JSP b.jsp. If one does not exist, or if one exists but is out of date, it will compile the JSP b.jsp contained in directory **/u/webusr1/sys02/web/b**.

Notes:

1. Invoking the jsp10BatchCompile.sh or jsp11BatchCompile.sh shell script to compile JSPs while the Application Server is running (and potentially serving these same compiled JSPs) WILL NOT RESULT in the Application Server recognizing that a JSP has changed and thus requires re-loading (in the case where reloading is enabled for that Web application). To indicate to the Application Server that a JSP has become outdated, you must modify the JSP's time stamp by changing (or touching) the JSP source file. When the Application Server notices that the time stamp for a JSP in a reloadable Web application is newer than the generated .class file, it automatically re-compiles and re-loads that JSP the next time it is requested.
2. Invoking the jsp10BatchCompile.sh or jsp11BatchCompile.sh shell script without parameters yields usage information.

Improving JSP compile time

Long JSP compile time and/or long javac compiler response time can result if the JAR files in the Web application classpath are lacking directory entries. (JAR files produced by VAJAVA and WSAD do not contain directory entries.) If you do not choose to pre-compile your JSPs, you might still be able to decrease the amount of time it takes to compile them.

To determine if a JAR file is missing directory entries, issue the following command:

```
 jar -tvfd bad.jar  > out.filenames
```

If the JAR file is includes directory entries, the resulting list of files will looked like this:

```
0 Tue Nov 12 11:36:30 EST 2002 it/
0 Tue Nov 12 11:36:30 EST 2002 it/customer1/
0 Tue Nov 12 11:36:30 EST 2002 it/customer1/ims/
0 Tue Nov 12 11:36:30 EST 2002 it/customer1/ims/utils/
```

If the JAR file is missing directory entries, the resulting list of files will looked like this:

```
1041 Tue Nov 12 11:36:30 EST 2002 it/customer1/ims/utils/DataRW.class
1867 Tue Nov 12 11:36:30 EST 2002 it/customer1/ims/utils/IMSBOManager.class
2513 Tue Nov 12 11:36:30 EST 2002 it/customer1/ims/utils/IMSConnectionManager.class
13473 Tue Nov 12 11:36:30 EST 2002 it/customer1/ims/utils/TrxExec.class
```

If the JAR file is not missing directory entries:

1. Place all of the classes to be included in the JAR file into a single directory.
2. Issue the following form of the jar command, where itdir is a directory containing the classes to be included in the JAR file:

   ```
   jar cvf WebApPo_utils.jar itdir
   ```

If directory entries are not contained in the output file, you can issue the following commands to rebuild the JAR file with directory entries:

```
mkdir good      (creates a clean directory)
cd good jar -xvf ../bad.jar     (extracts the classes into the new 'good' directory)
jar -cvf ../good.jar    (archives all the class files in the "good" directory)
```

and then:

1. Place all of the classes to be included in the JAR file into a single directory.

2. Issue the following form of the jar command, where itdir is a directory containing the classes to be included in the JAR file:

```
jar cvf WebApPo_utils.jar itdir
```

Compiling servlets

To compile a servlet:

1. Test that the appropriate Development Kit is in your system path by issuing one of the following commands:
   - java -version
   - java -fullversion

   Both of these commands should return a message stating the SDK version that is in your system path. If the command is not found, you will see an error message. See "Required Software Development Kit" on page 1-2 for the required Development Kit level for Version 3.5 of the Application Server.
2. Ensure that the system CLASSPATH environment variable includes your SDK rt.jar file and the appropriate Application Server JAR files. There are several JAR files in the *applicationserver_root***AppServer/lib** directory. Use the following table to determine which Application Server JAR files you will need.
3. Compile the classes by issuing the following command:

```
javac filename.java
```

API	**JAR file**
Java Servlet API 2.1	V2.2 servlet.jar
Java Servlet API 2.2	V2.2 servlet.jar
JavaServer Pages V0.91	ibmwebas.jar (but not material to compile)
JavaServer Pages V1.0	ibmwebas.jar (but not material to compile). ibmwebas.jar now includes code previously contained in jsp10.jar.
JavaServer Pages V1.1	ibmwebas.jar (but not material to compile)
PageLIstServlet	ibmwebas.jar (but not material to compile)
SessionsAPIs	ibmwebas.jar
Data access beans	databeans.jar
Connection Manager (deprecated)	ibmwebas.jar
Connection pooling	ibmwebas.jar
XML and XSL APIs	xml4j.jar and lotusxsl.jar

Passing init-parameters to a servlet

The init-parameters can be passed to a servlet:

- Through **webapp** properties in the was.conf file.
- Through XML tags in a <*webapp-name*>**.webapp** file. This file **must** includes a <servlet-path> tag.
- As part of the content of a .servlet file. An associated **webapp** property containing the **initargs** keyword must be included in the was.conf file to initialize the variable in the getInitParameter method contained in the servlet.

  Note: If you migrated from WebSphere Application Server V1.2, you might have used a continuation character ("\") on the **servlet.***servlet_name***.initargs**

property in the was.conf file to include additional variables for the getInitParameter method because a maximum of 255 characters were allowed on a single line in the was.conf file. Since up to 1024 characters are allowed on a single line in the V3.5 was.conf file, the use of this continuation charater is no longer supported.

Regardless of which method is used to pass the init-parameters to a servlet:

- The following line of code must be included in the servlet receiving the init-parameters:

  ```
  String s = getInitParameter("<parameter>")
  ```

  where <*parameter*> is the name of the parameter to be retrieved.
- If level 1.1 JSPs, which reside in the WEB-INF directory of the Web application document root, are to be compiled, either add the following property to the was.conf file for each affected Web application:

  ```
  webapp.<webapp-name>.servlet.jsp11.initargs=allowwebinf=true
  ```

  or or add the following tags to the appropriate <webapp-name>.webapp XML properties files:

  ```
  <init-parameter>
        <name>allowwebinf</name>
        <value>true</value>
  </init-parameter>
  ```

Using webapp properties in the was.conf file

When the Application Server receives a request for a Web application, it searches the was.conf file for **webapp.**<*webapp-name*> properties, where the name specified for <*webapp-name*> matches the name of the requested Web application. If it finds such properties, it uses them to configure the Web application's definition, including the init-parameters. Any .webapp file and/or .servlet file having the same name as the Web application's name, that exists in the same classpath as the Web application, will be ignored.

For example:

- There is a Web application called Payroll.
- This Web application invokes the servlet EmployeeServlet.
- The servlet EmployeeServlet is contained in the class file HRapps.
- The Web application and servlet are located in the classpath /u/Payroll/servlets
- parm1 is the name of the parameter to be retrieved.

If the following properties exist in the was.conf file, the Application Server will use these properties to define and initialize the EmployeeServlet servlet when the Payroll application is requested:

```
webapp.Payroll.servlet.EmployeeServlet=/servlet/*
webapp.Payroll.servlet.EmployeeServlet.code=HRapps
webapp.Payroll.servlet.EmployeeServlet.servletmapping=/servlet
webapp.Payroll.servlet.EmployeeServlet.initargs=parm1=HRapps.class
```

If a Payroll.webapp and/or Payroll.servlet file exist, their content will be ignored.

Using XML tags in a *<webapp-name>*.webapp file.

If a .webapp file is going to be used to pass an init-parameter to a servlet, no **webapp.**<*webapp-name*> properties defining that servlet can exist in the was.conf file. However the following **deployedwebapp.**property must be included to indicate where the invoking Web application is located:

```
deployedwebapp.<webappname>.classpath=<path>
```

Note: The associated .webapp file must be placed in the same classpath as the requested Web application.

For example, to specify the parm1 init-parameter in a .webapp file:

1. Add the following property to the was.conf file:
   ```
   deployedwebapp.Payroll.classpath=/u/Payroll/servlets
   ```
2. Include the following XML tags in the Payroll.webapp file that resides in the /u/Payroll/servlets classpath:
   ```
      <init-parameter>>
            <name>parm1</name>
            <value>500</value>
   ```

In the absense of **webapp.Payroll** properties that define the EmployeeServlet servlet, the Application Server will use the content of this Payroll.webapp file to configure the Web application and initialize the EmployeeServlet servlet. If a Payroll.servlet file also exists, it will be ignored.

Note: Regardless of the other XML tags the .webapp file contains, it MUST include a <servlet-path>tag defining where the servlet is located.

For more information about XML tags that can be included in a .webapp file to define a Web application, see the Deployment Descriptor Elements section in the Java Server Specification V2.2 at URL:

```
http://java.sun.com
```

Using a .servlet file

A .servlet file is an XML servlet configuration file and contains the name of the servlet class file, servlet initalization parameters, and a page list containing the URIs (universal resource identifiers) of the JSPs the servlet can call. The name of a .servlet file is derived from the name of the servlet for which it contains these configuration statements.

You can use a .servlet file to pass init-parameters to a servlet. However, the Application Server will only look for a .servlet file for the required servlet after it determines that **webapp** properties or a .webapp file are not available for the invoking Web application. The Application Server will then use the content of the .servlet file to define and initialize the servlet.

For example, to specify the parm1 init-parameter in a .servlet file, include the following init-parameter XML tag in the EmployeeServlet.servlet file:

```
   <init-parameter name="parm1" value="500"/>
```

For more information about XML tags that can be included in a .servlet file, see the *WebSphere Studio Guide* at URL:

```
http://www.ibm.com/software/websphere/studio/
```

Configuring servlet chaining

Servlet chaining refers to the process of satisfying a client request via a sequence of servlets, each servlet in the sequence piping its output to the next servlet, until the output of the last servlet in the chain is returned to the client. The chaining process is managed by the ChainerServlet, which invokes the chained servlets in order and manages the passing of data between them.

To use servlet chaining for a specific Web application:

1. Add the following property to the was.conf file to configure the invoker servlet:

   webapp.<*webapp_name*>**.servletmapping=/servlet/***

2. Add the following properties to the was.conf file to configure the ChainerServlet and specify the chained servlets as its init parameters:

   ```
   webapp.<webapp_name>servlet.<servlet_name>.servletmapping=/<servletmapping_name>
   webapp.<webapp_name>servlet.<servlet_name>.classpath=com.ibm.websphere.servlet.
        filter.ChainerServlet
   webapp.<webapp_name>servlet.<servlet_name>.initargs=chainer.pathlist=/servlet/
        <servlet1> /servlet/<servlet2> /servlet/<servlet3>
   ```

   The value specified for the **webapp.**<*webapp_name*>**servlet.**<*servlet_name*>**.servletmapping** property establishes an alias for the ChainerServlet that effectively represents the chain and through which the chain is invoked.

For example, to configure the Web application TestChain for servlet chaining where:

- Servlet chainServlet1 is the servlet being invoked.
- Servlet chainServlet2 is the second servlet in the chain.
- Servlet chainServlet3 is the last servlet in the chain and provides the response back to the client.

Add the following webapp properties to the was.conf file:

```
webapp.TestChain.servletmapping=/servlet/*
webapp.TestChain.servlet.CHAIN.code=com.ibm.websphere.servlet.
   filter.ChainerServlet
webapp.TestChain.servlet.CHAIN.servletmapping=/servletChain
webapp.TestChain.servlet.CHAIN.classpath=com.ibm.websphere.servlet.
   filter.ChainerServlet
webapp.TestChain.servlet.CHAIN.initargs=chainer.pathlist=/servlet/
   chainServlet1 /servlet/chainServlet2 /servlet/chainServlet3
```

Then to run this chain of servlets, you would invoke the ChainerServlet by entering the following URL from a browser:

```
 http://your.server.name/root-uri/servletChain
```

Servlet chaining can also be configured using XML tags in a <*webapp-name*>.webapp file or in a .servlet file. See the sections “Using XML tags in a <*webapp-name*>.webapp file.” on page 3-27 and “Using a .servlet file” on page 3-28 for more information.

To chain more than one set of servlets in the same Web application, configure the chainer servlet with a different sevlet names. For example, to configure the Web application TestChain with two servlet chains CHAIN1 (calling chainServlet10, chainServlet11, and chainServlet12) and CHAIN2 (calling chainServlet20, chainServlet21, and chainServlet22), add the following **webapp** properties to the was.conf file:

```
webapp.TestChain.servletmapping=/servlet/*
webapp.TestChain.servlet.CHAIN1.code=com.ibm.websphere.servlet.filter.
    ChainerServlet
webapp.TestChain.servlet.CHAIN1.servletmapping=/servletChain1
webapp.TestChain.servlet.CHAIN1.classpath=com.ibm.websphere.servlet.filter.
    ChainerServlet
webapp.TestChain.servlet.CHAIN1.initargs=chainer.pathlist=/servlet/
    chainServlet10 /servlet/chainServlet11 /servlet/chainServlet12
webapp.TestChain.servlet.CHAIN2.code=com.ibm.websphere.servlet.filter.
    ChainerServlet
webapp.TestChain.servlet.CHAIN2.servletmapping=/servletChain2
webapp.TestChain.servlet.CHAIN2.classpath=com.ibm.websphere.servlet.filter.
    ChainerServlet
webapp.TestChain.servlet.CHAIN2.initargs=chainer.pathlist=/servlet/
    chainServlet20 /servlet/chainServlet21 /servlet/chainServlet22
```

Then, to run the first chain of servlets, you would invoke the ChainerServlet by entering the following URL from a browser:

```
http://your.server.name/root-uri/servletChain1
```

To run the second chain of servlets, you would invoke the ChainerServlet by entering the following URL from a browser:

```
http://your.server.name/root-uri/servletChain2
```

Note: Each **webapp** property statement contained in the preceding examples must be entered as a single line in the was.conf file. Some of them are split here for printing purposes.

Chapter 4. Accessing relational databases

Servlets use the Java Database Connectivity (JDBC) API to access relational databases. Version 1.0 of the JDBC specification provided a base set of APIs for obtaining connections and subsequently driving SQL requests to relational databases. With the introduction of JDBC 2.0, the API has been split into:

- The JDBC 2.0 Core API, which contains evolutionary improvements, has been kept small and focused, like the JDBC 1.0 API, to promote ease of use. Code written for the 1.0 API continues to work with the 2.0 API. The 2.0 API classes remain in the java.sql package.
- The JDBC 2.0 Standard Extension API, which introduces additional capability in the programming and deployment of JDBC based applications. The programming model introduced by the JDBC 2.0 Standard Extension API is centered around the concept of a datasource. An instance of a datasource object contains attributes, such as a database URL, which describe the physical database (or data server) that is to be accessed at run time. In addition, data source objects expose interfaces that applications can use to create physical JDBC connections.

The JDBC 2.0 Specification also provides Java Naming and Directory Interface (JNDI) services which applications can use to locate datasource objects at run time. Previous versions of the Application Server contained support for defining JDBC database connection pools within the Application Server. This support introduced the notion of a Connection Manager and a set of programming APIs for servlets to use to access these pools at run time.

For compatibility, these APIs and capabilities continue to exist in this version of the Application Server. This version of the Application Server also provides an implementation of the JDBC 2.0 Standard Extension API for servlets to use. In particular, the Application Server provides an implementation of a datasource object and a JNDI initial context object. This implementation is able to take information from the Application Server was.conf file and create an in-memory directory structure that servlets can use at run time. This capability allows you to manage database related resources in a manner consistent with other Application Server defined resources.

The Application Server's implementation of the JDBC Standard Extension APIs is not dependent upon the level, nor the provider, of the JDBC Core API function.

This enables the Application Server to manage connections to databases that were created by the DB2 JDBC 1.0 Driver Manager, as well as connections to databases that were created with a JDBC 2.0 Core package and/or a comparable product provided by another vendor.

How servlets use the JDBC 2.0 Standard Extension API

The JDBC 2.0 Standard Extension API enables an application to use a datasource object to describe the physical database (or data server) that is to be accessed at run time. At run time, the datasource object acts as a factory for creating database connections. The application code is able to invoke a getConnection method on a datasource object which, in turn, obtains a JDBC connection handle that is returned to the application for use in accessing the database.

The JDBC 2.0 Standard Extension API Specification also uses the Java Naming and Directory Interface (JNDI) APIs to standardize how applications locate, and obtain a reference to, a datasource object. In particular, the specification suggests that applications use the Java Naming and Directory Interface (JNDI) APIs to obtain a naming context object that can subsequently be used to perform lookups into a Name Space. Whoever implements the naming context objects is responsible for using a logical name to register these implementations with the Java Naming and Directory Interface services. Applications can subsequently use the services provided by JNDI to produce a context at run time that has this logical name.

WebSphere Application Servers always register an initial context implementation under the name **com.ibm.ejs.ns.jndi.CNInitialContextFactory**. Applications use theses facilities to subsequently locate resources that have been registered by the Application Server as part of its initialization process. The following example illustrates the application code required to access a database using the JDBC Core and Standard Extension APIs. As the comment lines in this example indicate, some of the operations shown in the example should be performed initially, such as within the INIT method of a servlet, while others are intended to be used on a per request method, such as within the doGet method of a servlet.

```
/* APIs that should be invoked once per Servlet lifetime... */
Hashtable parms =new Hashtable();
parms.put(Context.INITIAL_CONTEXT_FACTORY,"com.ibm.ejs.ns.jndi.
    CNInitialContextFactory");
Context ctx =new InitialContext(parms);
DataSource ds =(DataSource)ctx.lookup("jdbc/sample");
/* APIs that should be invoked per request... */
conn =ds.getConnection("userid","password");
/* Sql requests against jdbc connection go here */
conn.close()
```

Notes:

1. Sometimes naming conflicts occur when using the com.ibm.ejs.ns.jndi.CNInitialContextFactory package. If a naming conflict occurs, use the com.ibm.ejs.ns.jndi.ws390.CNInitialContextFactory package instead of the com.ibm.ejs.ns.jndi.CNInitialContextFactory package. Both packages contain the same CNInitialContextFactory class.
2. The Application Server automatically loads the classes that implement both of the interfaces introduced by JNDI and the JDBC Version 2.0 Standard Extension APIs into the Application Server classpath as part of its startup procedure. Therefore, no action is required to enable these classes for use at run time. The Application Server does not provide an implementation of the JDBC Core Driver APIs that exist within the javax.sql package. You must obtain, and make available for use at run time, an implementation of the Core Driver APIs. For

example, you can obtain the JDBC driver provided by DB2 for use within the Application Server. For servlets to gain access to these services, they must have the following import statements in their source code:

```
import java.sql.*;
import com.ibm.ejs.dbm.jdbcext.*;
import javax.naming;
import javax.sql.*;
import java.util.Hastable;
```

Using JDBC 2.0 Standard Extension API with the Application Server

The Application Server provides an implementation of a datasource object for applications to use. The application code is insensitive to the processing that is employed within the datasource implementation for satisfying a getConnection request. That is, the application has no knowledge or dependencies on how a connection is established. The datasource may have gotten the connection from a pool of JDBC connections that it is maintaining, or may have been required to instantiate a new JDBC connection using the JDBC Core driver package that is configured on each request.

The Application Server's implementation of the datasource maintains connection pools within the Application Server. You can use properties in the was.conf file to manage these pools. Specifically, you can define a connection pool, stipulating things like the maximum number of connections that can be obtained, and then map a datasource to this pool for administrative purposes. For more information on configuring connection pools, see "Example of a JDBC connection pool definition" on page 4-8.

The JDBC datasource interface defines two forms of the getConnection method for obtaining a connection handle:

- One form of the getConnection method takes as input an explicit userid and password. The userid and password are used in conjunction with the Core JDBC API to produce a connection handle which has as its primary authorization ID the userid and password that was passed to it. JDBC Core Driver providers often have variances in their implementation of this support. For instance, the JDBC driver provided by the DB2 product ignores the userid and password passed on a JDBC connect request to its Driver Manager. Instead, it uses the security information obtained from the invocation thread as the primary authorization ID. To accommodate CORE Driver vendors who implement this support in different manners, the Application Server implementation of the getConnection(userid,password) request not only passes the userid and password explicitly to the Core Driver provider, but also establishes the identity represented by the userid and password on the execution thread prior to invoking the CORE Driver API. This ensures that Core Drivers that establish security in either of these manners can be used in conjunction with the Application Server extended API implementation.
- The second form provides a getConnection method which does not require any parameters (in contrast to the userid, password form). When this form of the API is used, the Application Server defaults to establishing the JDBC connection with a primary authorization ID equal to that of the hosting Web server.

The was.conf file provides properties for assigning a JNDI name to entities such as datasources. The Application Server, in turn, provides a lightweight implementation of a naming service. At startup, the Application Server constructs an in-memory name space from the information in the was.conf file. The Application Server also registers an implementation of a JNDI naming context with

JNDI Context Factory services. The Application Server also provides the **com.ibm.ejs.ns.jndi.CNInitialContextFactory** class as an implementation of a JNDI Context Factory. Applications can use this class as input to JNDI Factory services to produce an initial naming context that serves as the starting point for name resolution within the Application Server name space.

The Application Server provides naming service and naming context implementations that are not intended to support generalized, full function navigation over a hierarchal name space. Instead, the lookup() method on the context object is the only API supported for application use. All other APIs for this interface will throw a not-implemented exception if invoked. This restriction does not impose constraints on the JDBC programming model and does not inhibit application portability among WebSphere Version 3 Application Servers.

When navigating a name space, each access is relative to a context. It should also be recognized that the Application Server name space is implemented as a flat, name-value pair name space with no notion of such things as subcontexts and junctions. The name space implementation behaves like a hash table where items are registered by the full name specified in the was.conf file and subsequently fetched by applications using that name. The provision of an Application Server naming service does not prevent the use of other naming service implementations within the Application Server. The structure of the JNDI APIs allows multiple independent naming context implementations to be available. An application is able to access a specific implementation by specifying the desired provider's Context Factory implementation. The Application Server does not provide support for, or aid in configuring, datasources into a naming service other than the lightweight version provided with the Application Server.

See "Enabling communication with DB2" on page E-1 for additional information about setting up DB2 to communicate with the Application Server.

Setting up JDBC connection pools

Creating a connection to a database is a high overhead task that can significantly degrade mainline application performance. For this reason, an application often establishes pools of connections early in their processing and then serially reuses them on subsequent requests. In particular, it provides a mechanism for pooling Java Database Connectivity (JDBC) handles which represent physical connections to a database.

The JDBC driver implementation must be added to the Application Server classpath. JDBC pools are managed at the Application Server level. Even when use of a pool is constrained to a single application, it is not sufficient to include the database driver as part of the application's classpath because the Application Server may choose to maintain the connections within the pool beyond the life of that individual application. (See "Enabling communication with DB2" on page E-1 for additional informaiton on how to add the JDBC driver implementation to the Application Server classpath.)

In addition to providing efficient access to database connection handles, the Connection Manager also attempts to minimize the amount of excess resource that the system is using at any given point in time. The pooling properties contained in the was.conf file allow you to specify how the pool is to be managed by specifying settings such as:

- The JDBC database management system (DBMS) that is hosting the Application Server connection pools.

- The minimum number of connections that should be in the pool
- The maximum number of connections that can be obtained
- How long a request for a connection will wait until an exception is thrown
- The amount of time that a connection may be in-use before the pool manager considers the connection to be invalid
- The amount of time a connection can remain idle before the resource is returned to the database
- The identity with which JDBC connections will be established.

The Application Server attempts to manage these settings as efficiently as possible within the bounds you set and in accordance with real time usage and workload.

An Application Server instance can contain multiple connection pools for use by applications executing within that Application Server. A JDBC connection pool can be defined to the Application Server by adding the appropriate properties to the Application Server's was.conf file. Applications can then access the resources configured within a pool by either using the Connection Manager APIs that were provided with an earlier version of the Application Server, or by mapping a datasource to that pool.

Note: All of the Connection Manager APIs are deprecated in Version 3.5 and might not work in future releases. Therefore, new servlets should use the connection pooling model introduced in Version 3.02 instead of the Connection Manager.

If Connection Manager APIs are used, applications are not allowed to override values specified in the was.conf file. Any attempt by an application to override a value specified in the was.conf file will be ignored.

All connection pool properties contained in the was.conf file are of the form:

```
jdbcconnpool.<pool-name>.<property>=<value>
```

where <pool-name> is the name of the JDBC connection pool being defined (each connection pool within the Application Server must have a unique name); <property> is the property name; and <value> is the value for that property. In order to create a JDBC database connection pool, at least one property and non-null value must be specified for each <pool-name> you want to set up.

jdbcconnpool.<pool-name>.provider=DB2/OS390 | other

This property is used to specify the JDBC database management system (DBMS) that is hosting the Application Server connection pools. DB2/OS390, which indicates that DB2 is hosting the connection pools, is the default value for this property. Therefore, you only have to include this property if you are using a DBMS other than DB2.

jdbcconnpool.<pool-name>.jdbcdriver=<driver-class-name>

This property is used to specify the fully qualified name of the JDBC driver that will be used to interact with the physical database. This property is required if a datasource name is defined for this connection pool; otherwise, it is optional. If this property is specified, it will be used to limit the pool to connections that match the specified JDBC driver name. If a request doesn't match the pool's JDBC driver name, no connection will be obtained.

There can be only one driver associated with a pool.

jdbcconnpool.<pool-name>.databaseurl=<database-url>

This property is used to specify the database URL used for this connection pool. This property is required if a datasource name is defined for the pool; otherwise, it is optional. If this property is specified, it will be used to limit the pool to use by connections that match the specified database URL. If the request doesn't match the pool's database URL, a connection will not be made.

A pool definition can contain multiple versions of this property. When multiple database URLs are defined for a connection pool, the Application Server serves requests from that pool on a first come, first served basis. The entire set of connections, regardless of how they are distributed over multiple databases, are managed within the bounds of that pool's controls.

jdbcconnpool.<pool-name>.datasourcename=<name>

This property is used to specifies a datasource name for a connection pool. This property is required if the resources in the pool are to be exposed using a datasource implementation provided by the Application Server; otherwise, it does not need to be specified. The datasource name specified must be the exact name that an application will use when doing a context lookup. Only one datasource object can be mapped to a given connection pool.

jdbcconnpool.<pool-name>.connectionidentity=<string>

This property is used to specify the identity with which JDBC connections will be established. <string> can be one of the following values:

- connspec - The identity will be assigned from the userid field of the IBMJDBCConnSpec object.
- server - The identity will be that of the Web Server address space.
- thread - The identity will be that of the thread on which the JDBC Connection request is made.

If this property is not specified for a connection pool, the default value connspec is used.

jdbcconnpool.<pool-name>.maxconnections=<integer>

This property is used to specify the maximum number of connections that can exist within the connection pool. These connections can be in either an in-use or idle state. The idle state refers to a connection that is in the pool and available to be returned to a requesting client. In-use refers to a connection that has been returned to a client for use.

Once the maximum number of connections is reached, and all connections in the pool are in-use, subsequent requests for connections will either wait until a connection is released, or fail if the request will not tollerate waiting. If this property is not specified for a connection pool, the maximum number of connections the pool will have is 25.

jdbcconnpool.<pool-name>.minconnections=<integer>

This property is used to specify the minimum number of connections that can be in a pool after initial population. The pool manager does not initially populate the pool to contain this number of connections. Instead, the pool is populated on an on-demand basis. The pool manager will not remove idle connections from a pool if that action will result in the number of connections in the pool going below this minimum number. If this property is not specified for a connection pool, at least 1 connection will always remain in the pool.

jdbcconnpool.<pool-name>. waitforconnectiontimeoutmilliseconds=<time>

This property is used to specify how long, in milliseconds, a request for a connection will wait until an exception is thrown. This can happen when a request for a connection is received, the maximum number of connections has been met, there are no idle connections in the pool, and no in-use connections are returned to the pool prior to the timeout period.

Specifying a value of -1 for this property will result in a request for a connection failing immediately if no connection is available. It will not wait for a connection to become available.

If this property is not included for a connection pool, the pool manager will wait 30000 millisecs (30 seconds) before it will fail a connection request.

Note: This property must be entered in the was.conf file as a single line; it is split here for printing purposes.

jdbcconnpool.<pool-name>. inuseconnectiontimeoutmilliseconds=<time>

This property is used to specify how long, in milliseconds, a connection can be in-use before the pool manager invalidates it. This property is intended to guard against the situation where an application encounters a fatal error while holding a connection from the pool. When the pool manager determines a handle has exceeded this threshold, it will invalidate the connection and place it back in the pool for use by another request.

Independent of this support, applications have to be sensitive to receiving an invalid connection exception (i.e., the database has failed).

Specifying a value of -1 for this property will result in connections not being invalidated because the allotted connection time has been exceeded.

If this property is not included for a connection pool, the pool manager will wait 120000 millisecs, (120 seconds) before it will invalidate a connection and place it back into the pool.

Note: This property must be entered in the was.conf file as a single line; it is split here for printing purposes.

jdbcconnpool.<pool-name>. idleconnectiontimeoutmilliseconds=<time>

This property is used to specify the length of time that a database connection can remain idle (i.e. not used) in the pool before it is eligible to be removed, thus freeing up all the resourcesassociated with that connection.

If this property is not included for a connection pool, the pool manager will wait 120000 millisecs, (120 seconds) before it will remove an idle connection from the pool.

Note: This property must be entered in the was.conf file as a single line; it is split here for printing purposes.

Example of a JDBC connection pool definition

The following depicts the properties that might be included in the was.conf file to define the connection pool MyPool:

```
jdbcconnpool.MyPool.jdbcdriver=ibm.sql.DB2Driver
jdbcconnpool.MyPool.databaseurl=jdbc:db2os390:NETA
jdbcconnpool.MyPool.datasourcename=jdbc/sample
jdbcconnpool.<pool-name>.connectionidentity=connspec
jdbcconnpool.MyPool.minconnections=3
jdbcconnpool.MyPool.maxconnections=10
jdbcconnpool.MyPool.waitforconnectiontimeoutmilliseconds=30000
jdbcconnpool.MyPool.idleconnectiontimeoutmilliseconds=120000
jdbcconnpool.MyPool.inuseconnectiontimeoutmilliseconds=30000
```

Migrating Connection Manager Code to use the JDBC Standard 2.0 Extension APIs

This section uses code fragments to show how Connection Manager code (the ″old″ way) can be replaced with code that makes use of datasources (the ″new″ way.) Most servlets previously written to use the Connection Manager to access DB2 need to be updated with code similar to the following ″new″ code fragments:

- Make sure you have the necessary import statements. Consider dropping any unnecessary imports.

  Old (if migrating from V1.x):

  ```
  import java.sql.*;                   // for data server access     (keep)
  import com.ibm.servlet.connmgr.*;    // connection manager classes (drop)
  ```

  Old (if migrating from V3.02):

  ```
  import java.sql.*;                   // for data server access   (keep)
  import com.ibm.db2.jdbc.app.stdext.javax.sql.*; // IBM implementations.
      .. (delete)
  import com.ibm.ejs.dbm.jdbcext.*;    // ..of the extensions      (delete)
  import javax.naming.*;               // to get at naming service (keep)
  ```

  New:

  ```
  import java.sql.*;                   // for data server access   (retained)
  import javax.sql.*;                  // IBM implementations...   (new)
  import javax.naming.*;               // to get at naming service (retained)
  ```

- In the servlet init() method, if you are migrating from Version 1.x, do one-time initializations to establish variables for use by all client requests. The *spec*, *connMgr*, and *ds* variables shown below are actually instance variables and are not local to the init() method. Therefore, in actual servlets, these variables are first declared outside of all methods, and setting their values in init() would not be preceded with the class names IBMConnSpec, IBMConnMgr, and DataSource as shown in the following example. The class names are shown in this example to help you more easily identify the variables. For the ″new″ way, you must provide information on the arguments for the put() and lookup() methods.

  Old:

  ```
  // create specification for desired connection
  IBMConnSpec spec = new IBMJdbcConnSpec("poolname",
       true,
       "COM.ibm.db2.jdbc.app.DB2Driver",
       "jdbc:subprotocol:database",
       "userid",
  ```

```
     "password");
// establish connection manager access to use its facilities
IBMConnMgr connMgr = IBMConnMgrUtil.getIBMConnMgr();
```

New:

```
// create parameter list to access naming system
Hashtable parms = new Hashtable();
parms.put(Context.INITIAL_CONTEXT_FACTORY, "com.ibm.ejs.ns.jndi.
    CNInitialContextFactory");
// access naming system
Context ctx = new InitialContext(parms);
// get DataSource factory object from naming system
DataSource ds = (DataSource)ctx.lookup("jdbc/sample");
```

- For every client request, the doGet() or doPost() method will begin by getting a connection. The Connection Manager must be explicitly told, by the value specified on the spec variable (the ″old″ way), which connection pool to use. Using the ds DataSource variable, however, hides from the servlet programmer any need to be aware of connection pooling (the ″new″ way).

  Old:

```
 // use spec to get connection manager connection
IBMJdbcConn cmConn = (IBMJdbcConn)connMgr.getIBMConnection(spec);
// use connection manager connection to get data server connection
Connection conn = cmConn.getJdbcConnection();
```

  New

```
// use DataSource factory to get data server connection
Connection conn = ds.getConnection("userid", "password");
```

- For every client request, a connection is used to access DB2. This is usually in the doGet() or doPost() method. Since standard JDBC APIs are used for data access, no Connection Manager APIs are used. Therefore, no code changes need to be made. However, the JDBC Standard Extension APIs offer some new possibilities that you may want to consider for DB2 interaction.
- Near the end of every client request, usually in the doGet() or doPost() method, connection related resources should be released. However, there are subtle differences between how the ″old″ and ″new″ ways process the release of a connection.
  - Under the ″old″ way, you release the Connection Manager connection but you must not close the actual DB2 connection, since management of the DB2 connection must be done by the Connection Manager.
  - Under the ″new″ way, you actually invoke the close() method to release the DB2 connection itself. However, this does not really close the connection. Instead, it returns the connection to the connection pool for reuse.

  Old:

```
// release connection manager connection
cmConn.releaseIBMConnection();
// do not issue conn.close();
```

  New

```
// "close" connection, placing it back in the connection pool
conn.close();
```

It is possible, in rare situations, for a servlet using connection pooling to have its connection taken away. This is a ″preemption″ feature that is disabled by default, but which can be enabled through the JVM properties files. If preemption is enabled, a problem can arise if a single request makes multiple uses of the connection separated by extended periods of time. The connection might be considered an ″orphan″ connection owned by a servlet that has failed or otherwise become unresponsive. The connection pooling facility can take away an orphaned connection, returning it to the connection pool for reuse. The verifyIBMConnection() method can be included in servlet code to check for this case and to provide a means of recovery.

Servlets using the Connection Manager have the same problem. Therefore, any servlet that uses the verifyIBMConnection() method to check whether a connection has been preempted, must be migrated to check for a ConnectionPreemptedException and include a means of recovery when this exception is encountered.

Supported Connection Manager APIs

Some Connection Manager APIs are intended only for monitoring purposes or internal Connection Manager use, and do not have any practical use in production servlets. The Connection Manager classes (and associated methods) that can be used in production servlets are listed below:

- Class: com.ibm.servlet.connmgr.IBMConnMgrUtil

  Methods:

  ```
  Class: com.ibm.servlet.connmgr.IBMConnMgrUtil
  ```

  The next three methods are intended for WebSphere Studio use only.

  ```
  public static IBMConnPoolSpec getPoolProperties(String poolName)
  public static void addPoolProperties(IBMConnPoolSpec spec)
  public static String urlToPoolName(String url)
  ```

- Class: com.ibm.servlet.connmgr.IBMConnMgr

  Methods:

  ```
  public IBMConnection getIBMConnection(IBMConnSpec connSpec)
  public IBMConnection getIBMConnection(IBMConnSpec connSpec, String ownerClass)
  ```

- Class: com.ibm.servlet.connmgr.IBMConnection

  Methods:

  ```
  public boolean verifyIBMConnection()
  public void removeIBMConnection()
  public void releaseIBMConnection()
  ```

- Class: com.ibm.servlet.connmgr.IBMJdbcConn

  This class is derived from the IBMConnection class above and it implements the following additional method:

  ```
  public Connection getJdbcConnection()
  ```

- Class: com.ibm.servlet.connmgr.IBMConnPoolSpec

  This class and the associated methods are intended for WebSphere Studio use only.

  ```
  public IBMConnPoolSpec(String poolName,
                  String poolType,
                  int    maxConnections,
                  int    minConnections,
                  int    connectionTimeOut,
                  int    maxAge,
                  int    maxIdleTime,
                  int    reapTime)
  public IBMConnPoolSpec(String poolName, String poolType)
  ```

- Class: com.ibm.servlet.connmgr.IBMJdbcConnSpec

  Methods (the first three are constructors):

```
public IBMJdbcConnSpec(String  poolName,
               boolean waitRetry,
               String  dbDriver,
               String  url,
               String  loginUser,
               String  loginPasswd)
public IBMJdbcConnSpec(String poolName)
public IBMJdbcConnSpec()
public void verify()
```

Chapter 5. Session tracking

A session is a series of requests originating from the same user, at the same browser. Using the Application Server Version 3.5 implementation of the Java Servlet API session framework, your server can maintain state information about sessions.

The Application Server provides facilities we group under the heading Session Manager that support the javax.servlet.http.HttpSession interface described in the Servlet API specification. A session object can be implemented in a variety of ways, each of which provides different levels of performance, failover, and clustering. In all cases, the Application Server defines the notion of a session transaction. A session transaction begins when the servlet calls javax.servlet.http.HttpServletRequest.getSession(boolean). It ends with the completion of the servlet's javax.servlet.http.HttpServlet.service(request, response) method.

The Application Server, by default, locks sessions during the processing of a session transaction to maintain session integrity. This means that one, and only one, instance of a servlet can access a session at a given time. In the case where several servlets are chained together to service an individual http request, the session stays locked across each servlet in the chain until a response is finally sent back to the user.

The sync() method of the com.ibm.websphere.servlet.session.IBMSession interface (which extends the javax.servlet.http.HttpSession interface) can be used to override a session transaction. A servlet can call sync() while it is in its service() method to unlock the session transaction, and then call the getSession() method of the request object to re-lock it.

Session security

Maintaining the security of individual sessions is part of the overall security structure built into the Application Server. Unlike the 1.x versions of the Application Server, the servlet will no longer be able to set a user name associated with a session; that functionality is now incorporated into the Session Manager. When creating a session as part of request processing, the Application Server will use the value returned by the getRemoteUser method on the HTTP Request object as the user name associated with a session. If the getRemoteUser method returns null (which it will if a request does not require authentication), the Application Server uses a value of ″anonymous″ to denote the user. When processing a getSession() request on behalf of a Servlet, the Application Server validates that the user name associated with the current request matches the user name associated with the session. If the names do not match, the getSession method will throw an exception of com.ibm.websphere.servlet.session.UnauthorizedSessionRequestException.

User authentication is performed by the Web server prior to invoking the Application Server. The following table illustrates the different scenarios that may occur depending on whether the HTTP Request was authenticated and whether a valid session ID and user name were detected by the Session Manager.

	No session ID was passed in for this request, or an ID is passed in for a session that is no longer valid.	**A session ID for a valid session is passed in. The current session user name is ″anonymous″.**	**A Session ID for a valid session is passed in. The current session user name is ″FRED″.**	**A Session ID for a valid session is passed in. The current session user name is ″BOB″.**
Unauthen ticated HTTP request used to retrieve a session.	A new session is created and the user name is marked as ″anony- mous″.	The session is returned.	The session is not returned; Unauthori zedSession Request Exception is thrown.	The session is not returned; Unauthori zedSession Request Exception is thrown.
HTTP request authen- ticated, with an identity of ″FRED″ used to retrieve a session.	A new session is created and the user name is marked as ″FRED″.	The session is returned and the user name is changed by the Session Manager to ″FRED″.	The session is returned.	The session is not returned; Unauthori zedSession Request Exception is thrown.

Session state without cookies

When you first make a request for which session management is enabled, the HttpSession object is created and the session ID is sent to the browser as a cookie. On subsequent requests, the browser sends the session ID back as a cookie and the Session Manager uses it to find the HttpSession associated with this user.

There are situations in which cookies will not work. Some browsers do not support cookies. Other browsers allow the user to disable cookie support. In such cases, the Session Manager must resort to a second method, URL rewriting, to manage the user session. With URL rewriting, all links that you return to the browser or redirect have the session ID appended to them.

To use URL rewriting in the Application Server, you will need to enable URL rewriting in the Session Manager and use a servlet or a JSP to serve as an entry point. This entry point is not dependent on sessions for its processing; rather, it contains encoded HREFs to servlets in the application that are dependent on sessions. The following example shows how Java code may be embedded within a JSP:

```
<%
response.encodeURL ("/store/catalog") ;
%>
```

Note: If you want to use URL rewriting to maintain session state, do not include links to parts of your Web applications in plain HTML files (i.e., files with .html or .htm extensions). This restriction is necessary because URL encoding cannot be used in plain HTML files.

To maintain state using URL rewriting, every page that the user requests during the session must have code that can be understood by the Java interpreter. If your Web application and portions of the site that the user might access during the session contain plain HTML files, these files must be converted to JSPs. This will impact the application writer because, unlike maintaining sessions with cookies, maintaining sessions with URL rewriting requires that each servlet in the application use URL encoding for every HREF attribute on tags. Sessions will be lost if one or more servlets in an application do not call the encodeURL(String url) or encodeRedirectURL(String url) methods.

To rewrite the URLs that are returning to the browser, the servlet must call the encodeURL() method before sending the URL to the output stream. For example, if a servlet that does not use URL rewriting contains the code:

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

then this code must be replaced with:

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

To rewrite URLs that are redirecting, a servlet must call the encodeRedirectURL() method. For example, if a servlet contains the following code:

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

then this code must be replaced with:

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

The encodeURL() and encodeRedirectURL() methods are part of the HttpServletResponse object. In both cases, these calls check to see if URL rewriting is configured before encoding the URL. If it is not configured, it returns the original URL. Also, unlike previous releases, WebSphere no longer makes any checks to see if the browser making an http request has processed cookies, and thus halts encoding of the URL. If URL encoding is configured and response.encodeURL or encodeRedirectURL are called, the URL will be encoded.

Configuring session tracking

To activate the session tracking function within an Application Server instance, the appropriate properties must be added to the was.conf file that is specified during the Application Server initialization process. Following is an example of the properties that would need to be included in the was.conf file to enable non-persistent session support with an invalidation interval of 30 minutes (the value is specified in milliseconds). This example configures cookies as the mechanism for maintaining the state with the client.

```
session.enable=true
session.invalidationtime=1800000
session.cookies.enable=true
```

Note: This example illustrates a minimal set of options. The full set of session properties, including detailed descriptions, are provided in the default was.conf file, a copy of which is provide in Appendix B, “was.conf file template”, on page B-1.

Session data can be stored either in memory or in a DB2 table. Regardless of where this data is stored, if the reloading function is enabled, all sessions associated with a Web application will be destroyed whenever a servlet contained in that application is reloaded. These sessions will not destroyed if a JSP contained in that application is reloaded. (See “Class loading and optional reloading” on page 3-21 for more information about the reload function.)

Session clustering

To support propagating events across OS/390 nodes in a session cluster, the Application Server uses a database to track and manage sessions in the common pool of sessions across all OS/390 cluster nodes. With the use of a database as well as the general architectural changes implemented in this version of the Application

Server, the Application Server no longer maintains the notion of a session cluster client and a session cluster server. In a clustered environment, the session may be accessed on any one of the virtual hosts in a cluster; which one is actually accessed will be transparent to the end user.

During the processing of a session transaction, if the virtual host fails for whatever reason during the WebSphere HttpSession transaction, the update to the database does not occur, but the common pool of sessions remains functioning (including the session being processed during the failure, minus any updates made during the uncompleted transaction). For non-catastrophic failures (i.e., when the virtual host remains functional), any changes made to the session which cannot be completed are rolled back and the session reverts to its state prior to the start of the transaction. Otherwise, once the transaction is completed and the changes are committed, the session is still accessible regardless of the failure of an individual node.

Configuring a session cluster

The Application Server can be configured so that the hosting Web server session data can be accessed by instances of Web applications executing in the same or different Application Server instances. The Application Server instances hosting these Web applications may be executing within multiple Web server processes. The Web server processes may be located on the same or on a different OS/390 image. Essentially, a session cluster defines the scope in which the session data may be shared among Application Servers.

The Application Server uses a DB2 database as the sharing mechanism among Application Server instances. Any Version 3.02 or Version 3.5 Application Server that is executing on an OS/390 image and is able to access the central database is able to participate in the session cluster; Version 1.x Application Servers are **not** able to participate in the cluster.

To configure a session cluster, you must:

- Have your DB2 Administrator create a database table for use within the session cluster. For more information about creating DB2 tables see one of the following publications:
  - *DB2 for OS/390 V5 Administration Guide*, GC26–8957
  - *DB2 UDB for OS/390 V6 Administration Guide*, GC26–9003
  - *DB2 UDB for OS/390 and z/OS V7 Administration Guide*, SC26-9931

The table space in which the database table is created must be defined with row level locking (LOCKSIZE ROW). It should also have a page size that is large enough for the objects that will be stored in the table during a session. Following is an example of a table space definition with row level locking specified and a buffer pool page size of 32K:

```
CREATE TABLESPACE <tablespace_name>
        IN <database_name>
        USING STOGROUP <group_name>
             PRIQTY 52
             SECQTY 2
             ERASE NO
        LOCKSIZE ROW
        BUFFERPOOL BP32K
        CLOSE YES;
```

A DB2 table must then be defined within this table space for the Session Manager to use to process the session data. The following example shows the format of this table:

```
CREATE TABLE database_name.<table_name>
        (ID              VARCHAR(24)     NOT NULL,
        PROPID           VARCHAR(24)     NOT NULL,
        APPNAME          VARCHAR(32),
        LISTENERCNT      SMALLINT,
        EXPIRES          TIMESTAMP,
        LASTACCESS       TIMESTAMP,
        CREATIONTIME     TIMESTAMP,
        MAXINACTIVETIME  INTEGER,
        USERNAME         VARCHAR(256),
        SMALL            VARCHAR(3595)   FOR BIT DATA,
        MEDIUM           LONG VARCHAR    FOR BIT DATA
        )
IN DATABASE.<database_name>;
```

The DB2 Administrator must also create a type 2 unique index on the ID and PROPID columns of this table. The following is an example of the index definition:

```
CREATE TYPE 2 UNIQUE INDEX DATABASE.<database_name>.<index_name>
  ON DATABASE.<database_name>.<table_name>
  (ID , PROPID)
  USING STOGROUP <group_name>
   ERASE NO
  BUFFERPOOL BP0
  CLOSE YES;
```

Notes:

1. At run time, the Session Manager will access the target table using the identity of the hosting Web server. Any Application Server that is configured to use persistent sessions should be granted both read and update access to the subject database table.
2. HTTP session processing uses the index defined using the CREATE INDEX statement to avoid database deadlocks. In some situations, such as when the a relatively small table size is defined for the database, DB2 may decide not to use this index. When the index isn't used, database deadlocks can occur. If this situation occurs, see the DB2 Administration Guide for the version of DB2 you are using for recommedations on how to calculate the space required for an index, and adjust the size of the tables you are using accordingly.
3. It may be necessary to tune DB2 in order to make efficient use of the sessions database table and to avoid deadlocks when accessing it. Your DB2 Administrator should refer to the DB2 Administration Guide for specific information about tuning the version of DB2 you are using.

- Create a JDBC connection pool for the Session Manager to use at run time. The pool can be configured using normal connection pooling constructs. IBM recommends the following default characteristics for a JDBC Database Connection Pool definition for connection pools that are going to be used by the Session Manager:

```
session.dbjdbcpoolname=<session-pool_name>
jdbcconnpool.<session_pool -name>.JDBCConnectionPool.minconnections=10
jdbcconnpool.<session_pool_name>.maxconnections=40
jdbcconnpool.<session_pool_name>.inuseconnectiontimeoutmilliseconds=-1
jdbcconnpool.<session_pool_name>.jdbcdriver=ibm.sql.DB2Driver
jdbcconnpool.<session_pool_name>.databaseurl=your_db_url
jdbcconnpool.<session_pool_name>.connectionidentity=server
```

You can also add the following properties to the was.conf file to enable session persistence and to inform the Session Manager of the location of its entities:

```
session.enable=true
session.invalidationtime=1800000
session.cookies.enable=true
session.dbenable=true
session.dbjdbcpoolname=SessionPool
session.dbtablename=SessionDB
```

Notes:

1. The maximum number of connections allowed for the pool should be equal to the MaxActiveThreads value specified in your httpd.conf file for the Web server.
2. The minimum number of connections for the pool should be 1/4 of the maximum number of allowed connections.
3. Setting the inuseconnectiontimeout property to -1, disables the reclaiming of inuse connections for this pool.
4. The connectionidentity property must be set to server.
5. IBM recommends that you take the JDBC pool defaults for both the waitforconnectiontimeoutmilliseconds and idleconnectiontimeoutmilliseconds properties.
6. IBM recommends that the Session Manager should be the only user of this pool.

Session clustering considerations

You should be aware of the following caveats regarding how session management works within a clustered Web server environment:

- The definition of the putValue() method of the HttpSession interface in the current Java Servlet Version 2.1 and 2.2 API Specifications (as specified by Sun Microsystems) does not account for the possibility of a clustered environment. If you add an object to a session that does not implement the serializable interface, you do not have any way to propagate the object along with a given session (each session must be serialized across the cluster). Consequently, the object will not be sent to and from the database when session updates are made. To make your applications portable to a clustered environment, you must make any objects placed in a session serializable.
- When HttpSessionBindingListener and HttpSessionBindingEvent are used in a clustered Web server environment, the event will be fired in the Application Server where the session is currently being processed. This will occur in situations where:
  - The servlet explicitly invalidates the session.
  - The session times out.
  - A listener is removed from a session.
- Any changes to the database parameters require a restart of the associated Session Managers. Therefore, you must restart ALL instances of a Session Manager in a cluster. Session Management operates under the previous mode setting until you restart the Session Manager.

In-memory session pools

With Version 3.5, you can specify the number of in-memory sessions that are to be maintained. Once this number is surpassed, these functions are bypassed. General memory requirements for your hardware system, as well as your site's usage characteristics, will determine the optimum value for this number. Also, with larger numbers, you may need to increase the heap sizes of the java processes for the Application Servers.

If you do not wish to place a limit on the number of sessions maintained in memory and allow overflow, set the value contained in the base in-memory session pool size to true. Allowing for an unlimited amount of sessions, however, can potentially exhaust system memory and even allow for system sabotage (where somebody could write a malicious program that continually hits your site and creates sessions, but ignores any cookies or encoded URLs and never utilizes the same session from one http request to the next).

When overflow is not allowed, the Session Manager will still return a session with the HttpServletRequest's getSession(true) method if the memory limit has currently been reached, but it will be an invalid session which is not saved in any fashion. With the WebSphere extension to HttpSession, com.ibm.websphere.servlet.session.IBMSession, there is an isOverflow() method which will return "true" if the session is such an invalid session. Your application could then check this and react accordingly.

Appendix A. Migrating from previous Versions of the Application Server

The process for configuring and operating the Application Server is unchanged from Version 3.02 of the Standard Edition product. If you are migrating from Version 3.02, you should be able to configure the Version 3.5 Application Server by making a few small changes to your existing Version 3.02 was.conf files (see "Required changes if you are migrating from V3.02" on page A-5).

In addition to maintaining consistent configuration support, Version 3.5 of the Application Server provides a Servlet compatibility mode (see "Including Web components in a Web application" on page 3-5). In compatibility mode, the application server will not enforce portions of the Javasoft Servlet Version 2.2 Specification that are not compatible with the Version 2.1 Specification. Therefore, you are not required to make changes to existing servlets that were deployed in your Version 3.02 Application Server to accommodate changes imposed by the new specification level.

The Application Server continues to provide support for JavaServer Pages written to the 0.91 and 1.0 specification levels. The desired specification level must be specified within each Web application definition (see "Including Web components in a Web application" on page 3-5). Web applications that were deployed in version 3.02 of the Application Server that contain JavaServer Pages written to the 0.91 and 1.0 level do not have to be changed for this version of the Application Server.

Version 3.5 continues support for the following concepts introduced in Version 3.02 of the Application Server. These concepts and support levels are consistent with the support provided in Version 3.5 of the Application Server on MultiPlatforms.

- **Support for configuring virtual hosts and Web applications**. A Web application allows a group of Web components (servlets, JavaServer Pages, and static files, such as HTML and GIF files) to be grouped and managed as a single unit. A common practice in deploying a Web infrastructure is to define a single Web server to act as a host for multiple logical host names. For instance, a single Web server instance could serve requests for hosts www.mycompany.com and www.myothercompany.com. To support this capability, Web server configuration files allow resources to be defined specific to a particular host (i.e. Resource A, Resource B are part of host my.company.com).

  The Application Server now provides similar support. The was.conf file is used for configuring the Application Server and contains support for new properties that allow you to define virtual hosts. A virtual host is able to contain one or more Web applications.

  Grouping related components together is not a new concept for the Application Server. Previous versions of the Application Server allowed components to be implicitly grouped together. Components deployed on a particular Application Server shared a single servlet context and were administered as a single unit. However, there was no provision for managing the components at less than the Application Server level.

  The introduction of Web applications in this version of the Application Server does not prevent you from continuing to manage components at this level. You can achieve this effect by grouping existing components in a single Web application. "Deploying a Web application to the Application Server" on page 3-7 describes how to create a Web application that can be used to contain the servlets you currently have defined within your existing was.conf configuration. Similarly, steps are provided to show how to create a virtual host for this Web application.

- **Servlet API Specifications support**. The Application Server is able to host servlets that conform to the Servlet API Specification Version 2.2 or 2.1. Versions 1.x of the Application Server provided support for servlets written to the Servlet API 2.01 Specification level. The Servlet API 2.2 and 2.1 Specifications are NOT upwardly compatible from the Servlet API 2.01 Specification. Certain APIs have been changed or deprecated. Therefore, any servlet that uses one of these APIs must be updated to accommodate the changes. Servlets which do not utilize these APIs should be able to be deployed unchanged to this version of the Application Server. See "Migrating Servlets" on page A-7 for some of the more frequently used new and deprecated classes and methods.

  Servlets that have been developed and tested on WebSphere Application Server Version 3.5 on MultiPlatforms should be able to be deployed unchanged to this version of the OS/390 Application Server except where an explicit exception is noted. (See Appendix D, "Programming Model Restrictions", on page D-1 for a list of these explicit exceptions.)
- **JSP 0.91, 1.0 and 1.1 Specifications support**. The Application Server is able to host JavaServer Pages and JHTML files that comply with either the JavaServer Pages 0.91, 1.0, or 1.1 Specifications. JSPs written to the 0.91 Specification level are not upwardly compatible with level 1.0. See "Migrating JSPs" on page A-8 for migrating JSPs.

  Previous versions of the Application Server for OS/390 supported the 0.91 and 1.0 Specification levels. If a Web application includes JSPs that are being migrated from an earlier version of the Application Server for OS/390, a **webapp.**<*webapp-name*>**.jsplevel** property must be included in the Web application definition statements in the was.conf file to indicate to which level of the specification the Web application is compliant.
- **SDK 1.3 support**. The Application Server V3.5 run-time environment is built on SDK 1.3. You should be able to run most programs that ran under JDK 1.1x with little or no modification. The following list summarizes some minor potential incompatibilities:
  1. There are now two Timer classes:
     – java.util.Timer (new)
     – javax.swing.Timer (existed in V1.1x)

     If an application has the following two import statements:
     ```
     import java.util.*;
     import javax.swing.*;
     ```
     and refers to javax.swing.Timer by its unqualified name, the following import statement must be added in order for the ambiguous reference to class Timer to be correctly resolved:
     ```
     import javax.swing.Timer;
     ```
  2. The implementation of method **java.lang.Double.hashcode** has been changed to conform to the API specification. This change should not affect the behavior of existing applications because hashcode returns a truncated integer value.
  3. A new Permission class, **java.sql.SQLPermission**, has been added in version 1.3. WebSphere Application Server V3.5 on MultiPlatforms supports this new class; WebSphere Application Server for OS/390 V3.5 does not.

4. The internal implementation of the Java Sound APIs (in class **com.sun.media.sound.SimpleInputDevice**) now checks **javax.sound.sampled.AudioPermission**. This new check means that, under 1.3, applets must now be given the appropriate AudioPermission to access audio system resources
5. JInternalFrames are no longer visible by default. Developers must set the visibility of each JInternalFrame to true in order to have it show up on the screen.
6. The **TableColumn.getHeaderRenderer** method returns null by default. Therefore, you must use the new **JTableHeader.getDefaultRenderer** method instead to get the default header renderer.
7. The JTable's default text editor now gives setValueAt objects of the appropriate type, instead of always specifying strings. For example, if setValueAt is invoked for an Integer cell, then the value is specified as an Integer instead of a String. If you implemented a table model, you might have to change its **setValueAt** method to take the new data type into account. If you implemented a class that is used as a data type for cells, make sure that your class has a constructor that takes a single String argument.
8. It is no longer possible for sufficiently trusted code to modify final fields by first calling **Field.setAccessible(true)** and then calling **Field.set()**. An IllegalArgumentException will be thrown when an attempt is made to modify a final field. The JNI Set<Field> routines can be used to set non-static final fields.
9. The specification and behavior of the constructors **BasicPermission(String name)** and **BasicPermission(String name, String actions)** in class **java.security.BasicPermission** have been modified. When the name parameter is null, the constructors now throw a NullPointerException. When name is an empty string, the constructors now throw an IllegalArgumentException. This change of behavior is inherited by subclasses of **BasicPermission**. The change also affects the behavior of **java.lang.System.getProperty()** and **java.lang.System.setProperty()** whose implementations construct an instance of **PropertyPermission**, a subclass of **BasicPermission**. Because of this change, a call to **getProperty** or **setProperty** with an empty property name (that is, **getProperty("")** or **setProperty("", value)**) will result in an IllegalArgumentException. In previous SDK versions, such a call would return quietly with no exception.
10. The behavior of **java.net.URL** has changed for cases where a URL instance is constructed from a String. A final slash ('/') is not automatically added to a URL when the URL is constructed without one. For example, the following code:

    ```
    URL url = new URL("http://www.xxx.yyy");
    System.out.println(url.toString());
    ```

    now results in the following output:

    ```
    http://www.xxx.yyy
    ```
11. The javac complier has a new implementation with the following implications:
    – Inherited members of an enclosing class are now accessible.
    – A local variable or catch clause parameter can be hidden when it is declared within the scope of a like-named method parameter, local variable, or catch clause parameter.

- It is now illegal for a package to contain a class or interface type and a subpackage with the same name. A package, class, or interface is presumed to exist if there is a corresponding directory, source file, or class file accessible on the classpath or the sourcepath, regardless of its content.
- A qualified name in a constant expression must be of the form TypeName.identifier.
- Member classes of interfaces are inherited by implementing classes

12. **java.io.ObjectInputStream** has been optimized to buffer incoming data. This change should improve performance. This change causes ObjectInputStream to more frequently call the multi-byte read(byte[], int, int) method of the underlying stream. If underlying stream classes incorrectly implement this method, serialization failures may occur.
13. A public input method engine SPI as been included so that a client side adapter can be developed and distributed as a separate product and installed into any implementation of the Java 2 platform. Environments that are currently set up to allow text entry using a server-based input method should updated to use a different solution, such as host input methods.

For the most current Java for OS/390 documentation, go to URL:

`http://www.ibm.com/servers/eserver/zseries/software/java/`

- **was.conf configuration file support**. Application Server properties are contained in a single configuration file that is a refinement of the was.conf configuration file used in previous versions of the Application Server. While the single file concept remains, it is no longer necessary to physically represent the Application Server model using a nested directory structure. Therefore, the makeserver and updateproperties utilities used to propagate changes to this nested directory structure in Versions 1.1 and 1.2 have been eliminated.

  In order to execute applications that currently exist on Version 1.1 or Version 1.2 of the Application Server, you must:

  1. Create a was.conf file for this version of the Application Server.
  2. Make any necessary changes to your applications to accommodate the new Servlet API and JSP Specification levels.

In many cases, IBM extends the specification levels supported by the Application Server to provide additional features and customization options. If your existing applications use extensions from past Application Server releases, mandatory or optional migration might be necessary to utilize the same kinds of extensions in Version 3.5. From Version 3.0x to Version 3.5, the main migration areas concern the IBM extensions and the SDK. In contrast, migrating from Versions 1.1 and 1.2 requires updating applications with respect to the open specifications, such as the Java Servlet API. The following table summarizes potential migration areas.

Functional area	**Version 3.5.x support**	**Need to migrate from V3.0x?**	**Need to migrate from V1.x?**	**Details**
Servlets	Servlet 2.1 Specification and IBM extensions	No	Yes	Versions 1.1 and 1.2 supported the Servlet 2.01 Specification and IBM extensions that were updated in Version 3.0x.
Servlets	Servlet 2.2 Specification	No	Yes	See the Sun Microsystems Web site for information about the new Servlet 2.2 APIs.

Functional area	Version 3.5.x support	Need to migrate from V3.0x?	Need to migrate from V1.x?	Details
JSP files	Supported JSP specifications are V1.1 (recommended), V1.0 (supported), V0.91 (supported)	No**	Yes	Version 1.x supported only the JSP 0.91 Specification. ** If you did not already migrate JSP 0.91 files for use with Version 3.0x, small migration is required for use with Version 3.5. It is recommended you migrate to JSP 1.1, despite 0.91 support.
XML	XML 2.0.x supported; XML 1.1.x supported with restrictions	No***	No***	*** Migration from 1.1x to 2.0.x is not required, but you might decide to migrate based on criteria and 1.1.x restrictions described in "Migrating to XML API Version 2.0" on page A-10.
JDBC and IBM database connection support APIs	JDBC 2.0; connection pooling model	No	Yes	V1.2 supported JDBC 1.0 and a Connection Manager model. If still using Connection Manager in Version 3.0x, it is recommended you switch to connection pooling.
Sessions	IBM session support APIs	No	Yes	Need to migrate from V1.x deprecated classes, changes to clustering, URL encoding for use with V3.0x or V3.5.
Transactions	Java 1.2 transactions support	Yes	Yes	Version 3.0x provided proprietary IBM packages to simulate Java 1.2 functionality. Version 1.x did not provide any support. Migrate to Version 3.5 if your applications require this kind of support.

Migrating your existing configuration (was.conf file) settings

Required changes if you are migrating from V3.02

If you are migrating from Version 3.02 of the Application Server, you should only need to make one change to your existing was.conf file before using it with your Version 3.5 Application Server; you must change the value specified on the **appserver.version** property from **3.02** to **3.50**. All of the other property values you have specified in your existing was.conf file should produce the same results as they did in Version 3.02 when used in a Version 3.5 environment.

If you do not make any other changes to your existing Version 3.02 was.conf file, your Version 3.5 Application Server will be running in compatibility mode, which is the default value for the new **appserver.compliance.mode** property. Therefore, the Web applications you are running on this instance of the Application Server must **ALL** comply with the Java Servlet API 2.1 specification. If you want to run Web applications that comply with the Java Servlet API 2.2 specification, you must change the value specified for this property to **true**. (See "Maintaining compatibility with existing applications" on page 2-9 for more information about this new property.)

Required changes if you are migrating from V1.1 or V1.2

If you are migrating from Version 1.1 or 1.2 of the Application Server, many of the was.conf file properties remain unchanged for. These properties can be grouped as follows:

- **Server properties**. As with previous versions of the Application Server, you can define a classpath on an Application Server basis. Each Application Server instance maintains an application level classloader to locate and load the classes it needs. If you included utility classes, such as Connector Gateway and JDBC implementations, on your classpath specification for a previous version of the Application Server, you can add them to the classpath for this version of the Application Server.

  You can also continue to configure logging on an Application Server basis. Improvements have been made to the logging function in this version of the Application Server that result in simplified configurations. It is recommended that you use the default logging level and location when migrating to the new version.

  You are no longer required to control tracing. If tracing is required, IBM support will provide instructions for enabling it.
- **Session properties**. HTTP Session data can be shared across Version 3.x Application Servers . This function uses configuration properties which are included in the was.conf file. (See Appendix B, "was.conf file template", on page B-1 for a template of this file.) For purposes of migrating an existing configuration, no changes are required to your session configuration properties unless you want to share data across Version 3.x Application Servers. The properties used to define session characteristics in Versions 1.1 and 1.2 remain valid in this version of the Application Server.
- **Connection pool properties**. New Support for JDBC 2.0 Standard Extensions has been added to this version of the Application Server. If you use this new support, you will need to add additional configuration properties to your was.conf file. For migrating existing configurations, no changes are required as existing Connection Manager properties remain unchanged with one exception; the IBMConnMgr.JDBC.useSerVerIdentity property is no longer supported.
- **Logging properties**. The new **appserver.loglevel** and **appserver.logdirectory** properties replace the followingVersion 1.1 and 1.2 properties:
  - **log.error.level**
  - **log.error.destination**
  - **log.error.filename**
  - **log.event.level**
  - **log.event.destination**
  - **log.event.filename**
  - **log.time**

  .
- **Virtual host definitions**. For migration purposes, a single virtual host can be created.
- **Application definition**. For migration purposes, a single Web application can be created that includes all of the Web components used on a previous version of the Application Server.
- **JVM definition**. The JVM property in the was.conf file contains the fully qualified name of the file containing the run-time properties relating to the Java Virtual Machine. (JVM).) Appendix C, "default_global.properties file", on page C-1 provides a copy of the default version of this file that is shipped with the Application Server.

  The default settings for the properties contained in this file are based on internal benchmarking with the Application Server Version 3.5 and the Java Virtual Machine that is shipped with it. Therefore, IBM recommends that Application

Server instances be allowed to initialize with these default values until an explicit need to modify them is recognized.

See Appendix B, “was.conf file template”, on page B-1 for a copy of the was.conf file template that includes migration considerations for individual properties.

Migrating Web server directives and environment variable settings

As in previous versions of the Application Server, Version 3.5 is configured to the hosting Web server by including the appropriate ServerInit, ServerTerm, and Service directives in that Web server’s httpd.conf file. To configure this version of the Application Server, you need to update the existing directives in the httpd.conf file of the hosting Web server to point to the Application Server initialization, termination, and service routines.

The initialization routine no longer requires a *server_model_root* directory as input. Instead, this routine takes as input the fully qualified name of a was.conf file containing the Application Server’s configuration. The new format for these directives is described in detail in “Configuring a Web server to host an Application Server” on page 2-1.

The Application Server continues to rely on environment variables containing pointers to the SDK libraries. For information on required environment variables and how to set them, see “Configuring a Web server to host an Application Server” on page 2-1.

Migrating Web server directives

If you are migrating from Version 3.02, you do not have to make any changes to your Web server directives.

If you are migrating from Version 1.1 or 1.2 of the Application Server, and are updating a Web server httpd.conf file that contains existing Application Server directives, you **must** replace existing ServerInit, ServerTerm, and Service directives with corresponding directives containing the new format.

The new format for these directives for Version 3.5 is fully described in “Configuring a Web server to host an Application Server” on page 2-1.

Migrating Web server environment variables

Make sure that the JAVA_HOME environment variable and any other hardcoded pointers to the SDK libraries contained in the hosting Web server’s envvars file point to the SDK that is required by the Version 3.5 Application Server. (See “Required Software Development Kit” on page 1-2 for the supported SDK level.) For example, if you’ve installed the SDK in **/usr/lpp/java/J1.3.0**, before starting a hosting Web server, set **JAVA_HOME=/usr/lpp/java/J1.3.0/IBM/J1.3**

Migrating Servlets

If you have servlets that were developed using the Servlet API Specification Version 2.1, you do not need to modify them before running them on Version 3.5 of the Application Server. However, if you have servlets that were developed using a Servlet API Specification earlier than Version 2.1, you should modify those servlets to use Servlet API Specification 2.1 or higher.

The following table highlights a few of the classes and methods that were new or deprecated with Servlet API Specification 2.1. Refer to the Java Servlet API Specification for Versions 2.1 and 2.2 for complete information concerning new and deprecated APIs.

Note: An efficient way to check if a servlet is using APIs that have been changed in either the Version 2.1 or 2.2 Servlet API Specification is to compile that servlet using the Servlet API 2.2 libraries. The compile step will highlight any source code that may require changes. This version of the Application Server includes the libraries for Servlet API 2.2. See “Compiling servlets” on page 3-26 for details of how to compile a servlet using the Application Server libraries on OS/390.

Method or Class	**Status and recommendation**
RequestDispatcher	New. Use the forward method to forward a servlet response from one servlet to a second servlet for further processing. Use the include method to include part of the one servlet’s response in the body of another servlet’s response.
HttpSessionContext	Deprecated. See Session state for tips for sharing session information.
HttpSession. getSessionContext	Deprecated. For security reasons, no equivalent.
HttpSession. getMaxInactiveInterval	New. Sets the maximum time a session will be maintained by the servlet engine without a client request.
ServletRequest. getRealPath	Deprecated. Use ServletContext.getRealPath.
ServletContext. getServlet	Deprecated. Use ServletContext.getRequestDispatcher.
ServletContext. getResource	New. Use this method to obtain a servlet resource by requesting its URL.
ServletContext. getResourceAsStream	New. Use this method to obtain a servlet resource (as an InputStream) from its servlet context.
encodeUrl and encodeRedirectUrl methods of HttpServletResponse	Deprecated. But the fix is easy. Change Url to URL in the method names.
HttpSession.isRequested SessionIdFromUrl	Deprecated. Another easy fix. Change Url to URL in the method name.
HttpServiceRequest. setAttribute()	Deprecated. See Migrating JSP APIs for details.
HttpServiceResponse. callPage()	Deprecated. See Migrating JSP APIs for details.

Migrating JSPs

The following level 0.91 JSPs are deprecated in the Application Server Version 3.5:

- HttpServiceRequest.setAttribute()
- HttpSerivceResponse.callPage()

There are two options for migration:

- **Migrate to JSP 1.1:** It is recommended that you migrate JSPs run under previous versions of the Application Server, and develop new JSPs, to conform to the JSP 1.1 Specification. Refer to the Sun JSP 1.1 Specification for details.
- **Migrate servlets or JSPs that use HttpServiceRequest and HttpServiceResponse:** If your servlets or JSPs developed under the Application Server Version 1.x cast to methods of com.sun.server.http.HttpServiceRequest or com.sun.server.http.HttpServiceResponse, you must perform one of the following migration steps:

1. Migrate com.sun.server.http.HttpServiceRequest.setAttribute() to javax.servlet.http.HttpServletRequest.setAttribute(), and migrate com.sun.server.http.HttpServiceResponse.callPage() to javax.servlet.RequestDispatcher.
2. Recompile the servlets before you use them with Version 3.5. Recompiling is necessary because HttpServiceRequest and HttpServiceResponse are provided as interfaces (instead of classes) that are implemented by the Version 3.5 servlet container.

The following is an example of migrating HttpServiceResponse.callPage() to RequestDispatcher:

```
// Code example of using the old HttpServiceResponse.callPage()
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class UpdateJSPTest extends HttpServlet
{
   public void doGet (HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
    {
       String message = "This is a test";
       ((com.sun.server.http.HttpServiceRequest)req).
             setAttribute("message", message);
       ((com.sun.server.http.HttpServiceResponse)res).
             callPage("/Update.jsp", req);
    }
}
// Code example of using the new RequestDispatcher
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class UpdateJSPTest extends HttpServlet
{
   public void doGet (HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
    {
      String message = "This is a test";
      req.setAttribute("message", message);
      RequestDispatcher rd = getServletContext().
         getRequestDispatcher("/Update.jsp");
      rd.forward(req, res);
      //((com.sun.server.http.HttpServiceRequest)req).
         setAttribute("message", message);
      //((com.sun.server.http.HttpServiceResponse)res).
         callPage("/Update.jsp", req);
     }
}
```

Migrating servlets from Version 3.0x connection pooling to Version 3.5 connection pooling

Connection pooling (provided through datasource objects) was introduced in Version 3.0x. Version 3.0x servlets using connection pooling need to be changed slightly and recompiled to run under Version 3.5. You will need to change one import statement from this:

```
import
com.ibm.db2.jdbc.app.stdext.javax.sql.*;
```

to:

```
import javax.sql.*;
```

Also, an application component that obtains two or more connections to the same database manager, using either the same datasource or different datasources, must use datasources with JTA-enabled drivers. With the Version 3.02 Connection Manager, obtaining two connections to the same resource resulted in only one actual connection, allowing JDBC drivers to be used in this scenario.

Migrating servlets that use the Application Server Connection Manager

The Connection Manager APIs are deprecated in the Application Server Version 3.5 environment and might not work with releases beyond this one. Therefore, you should not write new servlets using the Connection Manager. New servlets should be written using the connection pooling model. (See "Setting up JDBC connection pools" on page 4-4 for a description of the connection pooling model.)

If existing applications contain servlets (or JSP files) that require database connections, and the servlets use the Connection Manager from Version 1.2, it is recommended that you update the servlets to use the new connection pooling model. For most servlets, this update consists of simple coding changes. Because you should not write new servlets using the Connection Manager, the details of Connection Manager coding are not described in this publication, except as needed in the migration. For servlets that are going to be migrated, the following functions need to be modified:

- Import statements
- Servlet init() methods
- How connections are obtained
- How data is accessed (JDBC APIs must be used)
- How connections are closed (conn.close() methods need to be added)

"Migrating Connection Manager Code to use the JDBC Standard 2.0 Extension APIs" on page 4-8 describes how to modify the import statements and the servlet init() methods, and how to close connections for the new connection pooling model.

Utilizing JDBC APIs for data access

A servlet employing sessions uses a database connection to access the data server for each client request. The access usually occurs in the doGet() or doPost() method. Standard JDBC APIs are used for data access. Because no Connection Manager APIs are used, no code migration is necessary. However, the JDBC 2.0 Standard Extension APIs offer some new possibilities to consider for data server interaction.

Migrating to XML API Version 2.0

If you have XML applications that use XML for Java API Version 1.1.x, you can use those applications with the Application Server Version 3.5 with certain restrictions, or you can migrate those applications to XML for Java API Version 2.0.x.

Most XML applications are partly based on the DOM or SAX APIs. The primary decision factor for migration to the API Version 2.0.x is whether the application must strictly adhere to the DOM or SAX APIs. If strict adherence is a requirement, use the API Version 2.0.x. If the applications employ some of the convenience features of the compatibility APIs, use the API Version 1.1.x. Also, there are some inherent performance improvements in the API Version 2.0.x APIs, but you can gain additional performance improvements by explicitly using non-validating parsers (instead of the API Version 1.1.x parsers, which force validation) in application environments where the data can be trusted.

You cannot mix Version 2.0.x APIs with Version 1.1.x APIs in the same XML application. Some of the API Version 1.1.x classes are deprecated or not supported in API 2.0.x The Application Server Version 3.5 XML4J.JAR file includes the following API Version 1.1.x packages for TX compatibility:

- com.ibm.xml.parser package
- com.ibm.xml.xpointer package

If you need parser function that is provided in the API Version 1.1.x but not in the API Version 2.0.x, you can use the Version 1.1.x classes for TX compatibility; use the API Version 1.1.x methods to create a parser, cause the parser to read input, and set options. The DOM tree returned by the TX compatibility classes is an instance of the TX classes from the API Version 1.1.x. The following table summarizes the methods of the API Version 1.1.x class com.ibm.xml.parser.Parser that are not supported or implemented in API Version 2.0.x.

Method	**Status**
Parser.addNoRequired AttributeHandler	Not supported. Throws java.lang.IllegalArgumentException.
Parser.getReaderBufferSize	Not supported. Throws java.lang.IllegalArgumentException.
Parser.setErrorNoByteMark	Not supported. Throws java.lang.IllegalArgumentException.
Parser.setReaderBufferSize	Not supported. Throws java.lang.IllegalArgumentException.
Parser.setProcess External DTD	Not implemented. Does not function the same as in the API Version 1.1.x.
Parser.setWarningNoDoctype Decl	Not implemented. Does not function the same as in the API Version 1.1.x.
Parser.setWarningNoXML Decl	Not implemented. Does not function the same as in the API Version 1.1.x.
Parser.stop	Not implemented. Does not function the same as in the API Version 1.1.x.

In the API Version 2.0.x, some Version 1.1.x methods are deprecated, as summarized by the following table.

Deprecated Method	**Recommendation**
com.ibm.xml.parser.Parent. addElement(Chil	Use appendChild().
com.ibm.xml.parser.EntityDecl .getName()	Use getNodeName().
com.ibm.xml.parser.TXNotation. getName()	Use getNodeName().
com.ibm.xml.parser.TXElement. getName()	Use getNodeName() or getTagName()
com.ibm.xml.parser.EntityDecl. getNDATAType()	Use getNotationName().
com.ibm.xml.parser.Namespace. getUniversalName()	See createExpandedName().
com.ibm.xml.parser.TXElement. getUniversalName()	Use createExpandedName().
com.ibm.xml.parser.TXAttribute.ok getUniversalName()	Use createExpandedName().
com.ibm.xml.parser.TXElement. isEmpty()	See hasChildNodes().
com.ibm.xml.parser.EntityDecl. isNDATAType()	This method will be removed in a future release.

Deprecated Method	Recommendation
com.ibm.xml.parser.TXAttribute. setAttribute(TXAttribute)	Use setAttributeNode().
com.ibm.xml.parser.TXNotation. setName(String)	This method will be removed in a future release.
com.ibm.xml.parser.DTD. setName(String)	This method will be removed in a future release.
com.ibm.xml.parser.TXText. splice(Element, int, int)	This method will be removed in a future release.

Appendix B. was.conf file template

Following is a copy of the Application Server was.conf file template. The template includes a description of the values that can be specified for the various properties in the file. It also includes property migration considerations which may be helpful if you are migrating from a previous version of the Application Server. The template is located in the *applicationserver_root*/**AppServer/properties/was.conf.template** file.

Note: Text following the number symbol (#) in column 1 is always treated as a comment, regardless of the property setting.

```
######################################################################
# (C) COPYRIGHT 2000-2001 IBM Corporation. All rights reserved.
#
#
#  Configuration file template for the IBM WebSphere Application Server
#  for OS/390 version 3.50.
#
#  The documentation in this file provides...
#
#  - Descriptions of the directives that are to be included in
#    the application server configuration file. For more information,
#    please read WebSphere Application Server Standard Edition: Planning,
#    Installing, and Using Version 3.50.
#
#  - Step by step details for defining a configuration file which
#    makes use of the environment (physical files, etc.) from a
#    previous version of the application server. That is, this
#    gives detailed instructions for updating this file to be
#    a working configuration file that maps over the enities in
#    your existing server_model_root structure. Please see
#    the Migration section below.
#
#
#  NOTES ON SYNTAX:
#
#  The property names consist of fixed portions (e.g. webapp)
#  and variable portions (e.g. <webapp-name>).  The fixed portions
#  must be in lowercase; the variable portion can be in
#  mixed case and is case sensitive.
#
#  In the following example..webapp, servlet and autostart are fixed
#  portions of the property name and must be in lowercase, while
#  <webapp-name> and <servlet-name> are variable portions within the
#  property name and can be specified in mixed case.
#
#  ex. webapp.<webapp-name>.servlet.<servlet-name>.autostart=true
#
#
# ================================================================== #
# ================================================================== #
#
#        DIRECTIVES GROUPINGS
#        =================
#        - Runtime Properties
#        - Http Session Tracking
#        - JDBC Database Connection Pool
#        - Virtual Host
#        - Web Application
#        - Servlet
#
```

```
#       Note: Throughout this file, <INSTALL_ROOT> refers to the
#           directory path of the mounted install image of the
#           product. The default is /usr/lpp/WebSphere.
#
#
# ================================================================== #
#
#   Runtime settings
#          - Version
#          - Classpath/Libpath/Path settings
#          - JVM settings
#          - Logging level & location
#          - Working Directory
#          - Servlet 2.2 Compliance mode
#
# ================================================================== #
#
#
#      appserver.version=3.50
#
#        Version number used to verify this is the correct version
#        of configuration file. The value is used by the Application
#        Server to validate the file contents and should not be
#        changed.
#
#        This property and value must not be deleted or changed.
#
#
appserver.version=3.50
#
#--------------------------------------------------------------------#
#
#      appserver.usesystemclasspath=true|false
#
#        If set to true, the current setting of the $CLASSPATH
#        environment variable will be appended to the generated
#        classpath.
#
#        The default is false.
#
#
#--------------------------------------------------------------------#
#
#      appserver.libpath=<librarypath>
#
#        The libpath specified will be appended to the generated
#        libpath in the Application Server.
#
#
#--------------------------------------------------------------------#
#
#      appserver.classpath=<classpath>
#
#        The classpath specified will be appended to the generated
#        classpath.
#
#
#--------------------------------------------------------------------#
#
#      appserver.name=<name>
#
#        Specifies the Application Server Name. This is used to
#        identify the Application Server in displays and log messages.
#
#        The default is "defaultServletEngine".
#
#--------------------------------------------------------------------#
```

```
#
#      appserver.jvmpropertiesfile=<fully-qualified-filename>
#
#        Specifies the fully qualified name of the properties file
#        that contains the JVM specific properties.
#
#        The default name is:
#        <INSTALL_ROOT>/AppServer/properties/default_global.properties
#
#
#---------------------------------------------------------------------#
#
#      appserver.loglevel INFO|ERROR|WARNING
#
#        Specifies the logging level of the Application Server.  The
#        recommended loglevel is WARNING.
#
#        The default is warning.
#
#
#---------------------------------------------------------------------#
#
#      appserver.logdirectory=directory_name | STDOUT
#
#        Specifies the directory that will contain the Application
#        Server log files.  This directory must exist and be writeable
#        to the Application Server.  If STDOUT is specified for this
#        property, the logging files will be written to STDOUT
#
#        The default is STDOUT.
#
#---------------------------------------------------------------------#
#
#      appserver.jspbasehrefadd
#
#        The value of this property is a boolean that
#        indicates whether a JSP will output the <base href>
#        tag when a JSP is invoked via callPage from a servlet.
#        Setting this property to false will disable the output
#        of the <base href> tag in the generated Java code of
#        a JSP for JSP's using the .91 JSP processor.
#
#        When this property value is false, the <base href> tag
#        can be manually added to JSPs to prevent the need
#        to specify full pathing for all references to items such
#        as beans.
#
#        The default is true.
#
#---------------------------------------------------------------------#
#
#      appserver.workingdirectory=<directory>
#
#        Specifies the directory that will be used by the Application
#        Server for temporary files, including the class files generated
#        by JSP compile processing. This should be a fully qualified
#        directory location. The default is
#           /tmp/WebSphere/AppServer/<appserver.name>
#        where <appserver.name> is the value of the appserver.name
#        property.
#
#---------------------------------------------------------------------#
#
#      appserver.nodetach=true|false
#
#        Specifies whether the http server worker thread will be
#        detached from the JVM on the completion of each individual
#        request.  The default value is 'false', which means the
#        thread will be detached after each request.
#
#----------------------------------------------------------------------#
#
#       appserver.permissions=
#
#       Specifies the UNIX style permission bits (rwxrwxrwx) which
#       will be used to set the owner/group/other permission bits
#       for the Application Server directories and files which are
#       created in the path defined by appserver.workingdirectory
#       and appserver.logdirectory, including the files generated
#       by JSP compile processing and the log files.
#
#       The default is 777.
#
#-----------------------------------------------------------------------#
#      appserver.compliance.mode=true|false
#
#        Specifies whether the servlet engine is running in full
#        compliance with the servlet 2.2 specification, or is running
#        in compatibility mode.  A value of 'true' indicates that the
#        server is running in full compliance mode, a value of 'false'
#        indicates the server is running in compatibility mode. See the
#        section "Maintaining compatibility with existing applications",
#        in Chapter 2 for a summary of the application processing
#        implications of running in compliance mode.
#
#        The default value is false.
#
#
#----------------------------------------------------------------------#
#
#      appserver.java.system.property=property.name=property.value
#
#        Specifies additional properties that can be passed directly
#        to the java virtual machine when the JVM initializes.  The
#        Application Server makes no attempt to validate or interpret
#        the properties or values.  Multiple instances of the
#        appserver.java.system.property can be specified in the
#        configuration file.
#
#        There is no default.
#
#----------------------------------------------------------------------#
#
#      appserver.java.extraparm=jvm_parm
#
#        Specifies additional JVM-specific parameters that can be
#        passed to the JVM on initialization.  These parameters are
#        not validated or interpreted by the Application Server, but
#        are passed directly to the JVM.  Note that incorrect values
#        for this property may cause the initialization of the JVM
#        to fail, which will cause the Application Server
#        initialization to fail.  Multiple instances of the
#        appserver.java.extraparm property can be specified.  Only
#        one JVM parameter per property instance can be specified.
#        It is recommended that this property only be used under
#        guidance from IBM support.
#
#        There is no default.
#
#----------------------------------------------------------------------#
#
#      appserver.configviewer=<root-URI>
#
#        Specifies the URI root for the configuration viewer
#        which is automatically configured into each virtual host
#        within the Application Server.
#
#        The default is /ConfigViewer  which means that you would
#        specify a URL of <hostname>/ConfigViewer/showCfg  to access
#        the configuration viewer.
#
#---------------------------------------------------------------------#
#
#      appserver.initializeonwebappfailure=true|false
#
#       Specifies the initialization of AppServer when one or
#       more WebApp fails to load. When the property is set to true,
#       AppServer initializes if atleast one webapp loads successfully.
#
#       The default value is false. If property is set to false
#       Appserver fails to initialize if atleast one webapp fails.
#
#       If all the WebApps are loaded successfully, AppServer
#       initializes regardless of the value set through the property.
#
#---------------------------------------------------------------------#
#
#      objectleveltrace.enabled=true|false
#
#        Specifies whether object level trace support is enabled.
#
#        The default value is false. When value is set to true, you
#        must also set next two properties:
#
#      objectleveltrace.host=<host_name>
#
#        Specifies the Object Level Trace/Distributed Debugger
#        application host name or its IP address.
#
#      objectleveltrace.port=<port_number>
#
#        Specifies the object level trace application port number.
#
#---------------------------------------------------------------------#
#
#      inline.comment=true|false
#
#        Specifies whether the '#' character is considered to be
#        comment or data on subsequent properties.  The default
#        is 'false', which means that all data following the '#'
#        character is considered comment.  A value of 'true' means
#        that a '#' character found anywhere outside column 1 is
#        considered data.
#
#        The behavior of this property depends upon where in the
#        configuration file it is found.  When detected, this
#        property affects the parsing of lines which follow it.
#        The behavior of this property remains in effect for
#        subsequent properties unless it is specifically disabled.
#
#        The property may be toggled, for example:
#
#        inline.comment=true
#        webapp.myApp.servlet.theWebApp.code=myWebApp
#        webapp.myApp.servlet.theWebApp.initargs=param=param1#param2
#        inline.comment=false
#
#        The default is false.
#
# =================================================================== #
#
```

```
#   Session Settings
#
# ================================================================== #
#
#      session.enable=true|false
#
#        The value of this property is a boolean that
#        indicates whether session tracking is enabled. If
#        the property is set to "true," the session-related
#        methods for the request and response objects will
#        be functional.
#
#        If session is disabled and an application within the
#        Application Server attempts to use the session services,
#        an exception will be thrown.
#
#        The default is true.
#
#
#
#--------------------------------------------------------------------#
#
#      session.urlrewriting.enable=true|false
#
#        The value of this property is a boolean that
#        indicates whether session tracking uses rewritten
#        URLs to carry the session IDs. If the property is
#        set to "true", the Session Tracker recognizes
#        session IDs that arrive in the URL and rewrites
#        the URL, if necessary, to send the session IDs.
#
#        The default is false.
#
#
#--------------------------------------------------------------------#
#
#      session.cookies.enable=true|false
#
#        The value of this property is a boolean that
#        indicates whether session tracking uses cookies to
#        carry the session IDs. If the property is set to
#        "true", session tracking recognizes session IDs that
#        arrive as cookies and tries to use cookies as a means
#        for sending the session IDs.
#
#        The default is true.
#
#
#--------------------------------------------------------------------#
#
#      session.protocolswitchrewriting.enable=true|false
#
#        The value of this property is a boolean that
#        indicates whether the session ID is added to a URL
#        when the URL requires a switch from HTTP to HTTPS, or
#        HTTPS to HTTP.
#
#        The default is false.
#
#
#--------------------------------------------------------------------#
#
#      session.cookie.name
#
#        The value of this property is a string that specifies
#        the name of the cookie, if cookies are enabled.  The
#        cookie name must only contain:
```

```
#              -English alphanumeric characters (uppercase or
#                    lowercase A to Z and numbers 0 to 9)
#              -Period (.)
#              -Underscore (_)
#              -Hyphen (-)
#
#
#        The initial setting is "sesessionid".
#
session.cookie.name=sesessionid
#--------------------------------------------------------------------#
#
#      session.cookie.comment=<comment>
#
#        The value of this property is a string that specifies
#        a comment about the cookie, if cookies are enabled.
#
#        The default is "WebSphere Session Support".
#
#
#--------------------------------------------------------------------#
#
#      session.cookie.maxage=<integer>
#
#        The value of this property is an integer that
#        specifies the amount of time, in milliseconds, that a
#        cookie will remain valid. Specify a value only to
#        restrict or extend how long the session cookie will
#        live on the client browser.
#
#        By default, the cookie only persists for the current
#        invocation of the browser. When the browser is shut down,
#        the cookie is deleted.
#
#        The default is -1.
#
#
#--------------------------------------------------------------------#
#
#      session.cookie.path=<path>
#
#        The value of this property is a string that specifies
#        the path field that will be sent for session cookies.
#        Specify a value only to restrict to which paths on the
#        server (and, therefore, to which servlets, JHTML files,
#        and HTML files) the cookies will be sent.
#
#        Specifying "/" for the path indicates the root directory,
#        which means that the cookie will be sent on any access to
#        the given server.
#
#        The initial setting is "/".
#
session.cookie.path=/
#
#--------------------------------------------------------------------#
#
#      session.cookie.secure=true|false
#
#        The value of this property is a boolean that
#        indicates whether session cookies include the secure
#        field. If this property is set to "true", this will
#        restrict the exchange of cookies to only HTTPS
#        sessions. Otherwise, they will be exchanged in
#        HTTP sessions as well.
#
#        The default is false.
```

```
#
#
#---------------------------------------------------------------------#
#
#      session.cookie.domain=<domain-name>
#
#        Specifies the domain name for which the session cookie is
#        valid.
#
#        The default is null.
#
#
#---------------------------------------------------------------------#
#
#      session.invalidationtime=<milliseconds>
#
#        The value of this property is an integer that
#        specifies the amount of time in, milliseconds, that a
#        session is allowed to go unused before it is no
#        longer considered valid.
#
#        The default is 180000 millisecs, or 180 seconds.
#
#
#---------------------------------------------------------------------#
#
#      session.tableoverflowenable=true|false
#
#        Specifies whether there is a limit on the number of session
#        objects that should be maintained by the Application Server,
#        or whether the number of session objects that should be
#        maintained is unlimited. The number of session objects
#        is controlled by the session.tablesize property.
#
#        The default value is true, which means that the number
#        of session objects is unlimited.
#
#
#
#---------------------------------------------------------------------#
#
#      session.tablesize=<integer>
#
#        Specifies the size of the session table used to maintain
#        session objects within the Application Server. When
#        session.tableoverflowenable=false, this is the limit on
#        the number of session objects that can be created by the
#        Application Server at any one time.  When
#        session.tableoverflowenable=true, this represents the
#        initial size of the session table and the quantity by
#        which it is expanded.
#
#        The default is 1000 session objects.
#
#
#---------------------------------------------------------------------#
#
#      session.dbenable=true|false
#
#        Specifies whether or not the session objects should be stored
#        in a database.
#
#        The default value is false, which means that the session
#        objects are stored using memory in the JVM of the Application
#        Server instance that created the session.
#
#
```

```
#--------------------------------------------------------------------#
#
#      session.dbjdbcpoolname=<session-jdbc-poolname>
#
#        Specifies the name of the JDBC Database Connection Pool name
#        for use by the session support whenever
#        session.dbenable=true
#
#        IBM recommends the following default characteristics for a
#        JDBC Database Connection Pool definition for use by the
#        session services:
#
#   jdbcconnpool.SessionJDBCConnectionPool.minconnections=10
#   jdbcconnpool.SessionJDBCConnectionPool.maxconnections=40
#   jdbcconnpool.SessionJDBCConnectionPool.
#                                inuseconnectiontimeoutmilliseconds=-1
#   jdbcconnpool.SessionJDBCConnectionPool.jdbcdriver=ibm.sql.DB2Driver
#   jdbcconnpool.SessionJDBCConnectionPool.databaseurl=your_db_url
#   jdbcconnpool.SessionJDBCConnectionPool.connectionidentity=server
#
#        The pool name, SessionJDBCConnectionPool, is just an example.
#          Whatever name is used must match the value specified
#          on the  session.dbdbcpoolname property.
#        The pool properties have the following characteristics:
#        - maxconnections for the pool should be equal to the
#          MaxActiveThreads value in your httpd.conf file for the web server.
#        - minconnections for the pool should be 1/4 of the maxconnections.
#        - inuseconnectiontimeout should be set to -1, which disables
#          the reclaiming of inuse connections for this pool.
#        - jdbcdriver must be the DB2 jdbc driver
#        - databaseurl must be the URL of the target database
#        - connectionidentity must be server
#        IBM recommends that you take the JDBC pool defaults for both
#        - waitforconnectiontimeoutmilliseconds
#        - idleconnectiontimeoutmilliseconds
#        - datasourcename (not used by the session services)
#        IBM recommends that the session services should be the exclusive
#        user of this pool.
#
#        There is no default.
#
#
#--------------------------------------------------------------------#
#
#      session.dbtablename=<database-tablename>
#
#        Specifies the database table name to be used by the session
#        services when session.dbenable=true.
#
#        There is no default.
#
#
# ================================================================== #
#
#   JDBC Database Connection Pool Settings
#
#      You can define one or more JDBC Database Connection Pools per
#      Application Server. The syntax of the property name is:
#
#          jdbcconnpool.<pool-name>.<property>=<value>
#
#      where <pool-name> is the name of the JDBC Database Connection Pool
#            <property> is the property name
#            <value> is the value for the property
#
#      To create a JDBC Database Connection Pool, at least
#      one property and one non-null value must be specified.
```

```
#
#      The following properties exist for each connection pool:
#
#--------------------------------------------------------------------#
#
#      jdbcconnpool.<pool-name>.minconnections=<integer>
#
#        Specifies the minimum number of connections for the pool.
#        The pool is not initialized with this number of connections;
#        however once this number is reached, it represents the
#        minimum number of connections that should be kept in the
#        pool.
#
#        The default is 1.
#
#--------------------------------------------------------------------#
#
#      jdbcconnpool.<pool-name>.maxconnections=<integer>
#
#        Specifies the maximum number of connections for this pool.
#        Once the maximum number of connections is reached and all
#        connections in the pool are in-use, subsequent requests from
#        the pool will either be waited or failed based upon whether
#        the request will tolerate waiting.
#
#        The default is 25.
#
#--------------------------------------------------------------------#
#
#      jdbcconnpool.<pool-name>.
#                  waitforconnectiontimeoutmilliseconds=<milliseconds>
#
#        This should be on one line; it is split here because of
#        spacing constraints.
#
#        Specifies the wait, in milliseconds, for the connection timeout
#        value for this pool.
#        When all the connections in the pool are inuse, subsequent
#        requests from the pool will wait, up to the timelimit defined
#        by this property, for a connection to be released to the pool.
#        If no connection is released in the timelimit specified,
#        the request is failed.
#        If -1 is specified, it disables waiting for connections.
#        Hence, any request for a connection from the pool when all the
#        connections in the pool are already in-use will be failed
#        immediately, without waiting for connections to be released.
#
#        The default is 30000 millisecs or 30 seconds.
#
#
#--------------------------------------------------------------------#
#
#      jdbcconnpool.<pool-name>.
#                  idleconnectiontimeoutmilliseconds=<milliseconds>
#
#        This should be on one line; it is split here because of
#        spacing constraints.
#
#        Specifies, in milliseconds, the idle connection timeout for
#        this pool.
#        This specifies the length of time that a database connection
#        can remain idle (i.e. not used) in the pool before it is
#        eligible for removal, thus freeing up all the resources
#        associated with the database connection.
#
#        The default is 120000 millisecs or 120 seconds.
#
```

```
#--------------------------------------------------------------------#
#
#      jdbcconnpool.<pool-name>.
#                  inuseconnectiontimeoutmilliseconds=<milliseconds>
#
#        This should be on one line; it is split here because of
#        spacing constraints.
#
#        Specifies, in milliseconds, the in-use connection timeout for
#        this pool.
#        This specifies the length of time that a database connection
#        can be out of the pool before it is eligible for reclaiming
#        by the connection pool manager. This function guards against
#        an application that obtains a connection from the pool, but
#        does not return it within the timelimit defined by this
#        property.
#        If -1 is specified, it disables in-use connection processing
#        for this pool.
#
#        The default is 120000 millisecs, or 120 seconds.
#
#--------------------------------------------------------------------#
#
#      jdbcconnpool.<pool-name>.jdbcdriver=<driver-class-name>
#
#        Specifies the JDBC driver used for this pool. This is
#        required if a datasource name is defined for the pool.
#        Otherwise, it is optional and if specified, will be used to
#        constrain the pool to connections that match the specified
#        JDBC driver name.  If the request doesn't match the pool's
#        JDBC driver name, it will be failed.
#
#        The default is null.
#
#--------------------------------------------------------------------#
#
#      jdbcconnpool.<pool-name>.databaseurl=<database-url>
#
#        Specifies the database URL used for this pool. This is
#        required if a datasource name is defined for the pool.
#        Otherwise, it is optional and if specified, will be used to
#        constrain the pool to connections that match the specified
#        database URL.  If the request doesn't match the pool's
#        database URL, it will be failed.
#
#        The default is null.
#
#--------------------------------------------------------------------#
#
#      jdbcconnpool.<pool-name>.datasourcename=<name>
#
#        Specifies a datasource name for this connection pool.
#        This is required if the Connection Pooling APIs are going
#        to be used to obtain connections from this pool. Otherwise,
#        it does not need to be specified.
#
#        The name specified should be the same name that the your
#        application will use to perform the naming service lookup
#        on the datasource object.
#
#        The default is null.
#
#--------------------------------------------------------------------#
#
#      jdbcconnpool.<pool-name>.connectionidentity=<string>
#
#        Specifies the identity with which JDBC connections will
#       be established. <string> can be one of the following:
#
#       connspec - The identity will be assigned from the userid
#                  field of the IBMJDBCConnSpec object.
#       server   - The identity will be that of the Web Server
#                  address space.
#       thread   - The identity will be that of the thread on
#                  which the JDBC Connection request is made.
#
#       The default value is connspec.
#
#--------------------------------------------------------------------#
#
#     jdbcconnpool.<pool-name>.provider= DB2/OS390 | other
#
#       Specifies the JDBC database host:
#
#       DB2/OS390 : the DBMS is DB2 running on OS/390 or z/OS.
#       other     : the DBMS is not DB2 on OS/390 and no DB2
#                   specfic optimizations should be used.
#
#       The default is DB2/OS390
#
# ================================================================== #
#
#   Virtual Host settings
#
#     You can define one or more Virtual Hosts per Application Server.
#     The syntax of the property name is:
#
#         host.<virtual-hostname>.<property>=<value>
#
#     where <virtual-hostname> is the name of the Virtual Host
#           <property> is the property name
#           <value> is the value for the property
#
#     The following properties exist for each virtual host:
#
# ================================================================== #
#
#     host.<virtual-hostname>.alias=<hostname>|localhost
#
#       Specifies a hostname alias to be associated with this virtual
#       host name. This property provides a binding between the
#       hostnames understood by the web server and the virtual host
#       definitions in the Application Server.
#       There can be multiple alias properties per virtual host.
#
#       The application server supports a special hostname, "localhost",
#       which maps all requests to the virtual host definition.
#       This support is provided for the initial verification program.
#       IBM recommends that it not be used beyond that purpose.
#
#       There is no default.
#
#
#--------------------------------------------------------------------#
#
#     host.<virtual_hostname>.mimetypefile=<fully-qualified-filename>
#
#       Specifies the fully qualified filename of the mimetype properties
#       file used for this virtual host.
#
#       The default is:
#       <INSTALL_ROOT>/AppServer/properties/default_mimetype.properties
#
# ================================================================== #
```

```
#
#   Web Application Settings
#
#   A Web Application is made up of two sets of properties.
#
#   - Deployed Web Application properties
#     These properties represent characteristics that are unique to
#     the environment in which the web application is deployed.
#
#   - Web Application properties
#     These properties represent characteristics of the
#     content that comprises the web application.
#
#     One or more web application properties are required unless the
#     application's component parts are defined in a
#     <webapp-name>.webapp XML file. If so, only deployed web application
#     properties should be defined for the web application.  The
#     Application Server will search the class path to find the
#     <webapp-name>.webapp file.
#
# ================================================================== #
#
#   Deployed Web Application Properties
#
#     These properties represent characteristics that are unique to
#     the environment in which the web application is deployed.
#     These properties have the following syntax:
#
#      deployedwebapp.<webapp-name>.<property>=<value>
#
#      where <webapp-name> is the name of the web application
#            <property> is the property name
#            <value> is the value for the property
#
#      The deployed web application properties are:
#
#--------------------------------------------------------------------#
#
#      deployedwebapp.<webapp_name>.host=<virtual-hostname>
#
#      Defines the name of the virtual host into which this
#      web application is being deployed. This property is required.
#
#      There is no default.
#
#--------------------------------------------------------------------#
#
#      deployedwebapp.<webapp-name>.rooturi=
#
#      Defines the root URI for this web application. This defines
#      a pattern by which the web application is known within the
#      virtual host. This property is required.
#
#      There is no default.
#
#--------------------------------------------------------------------#
#
#      deployedwebapp.<webapp-name>.classpath=
#
#      This property specifies the classpath that the application
#      level class loader uses to searche for classes when the system
#      class loader cannot locate the class. This property is required.
#
#      There is no default.
#
#--------------------------------------------------------------------#
#
```

```
#      deployedwebapp.<webapp-name>.documentroot=
#
#      This property is used to specify the fully qualified name
#      of a directory containing JSPs, JHTML and static content to
#      be served by the Application Server. This property is required.
#
#      There is no default.
#
#--------------------------------------------------------------------#
#
#      deployedwebapp.<webapp-name>.authresource.<resource-name>=
#                                                <servletmapping>
#
#      This should be on one line; it is split here because of
#      spacing constraints.
#
#      This property is used to identify a web resource so that
#      access control policies can be applied to them.
#
#      <resource-name> is the resource name that is to be used
#      along with the virtual-hostname and webapp-name to construct
#      the SAF resource name of the form:
#         <virtual-hostname>.<webapp-name>.<resource-name>
#
#      <servletmappping> is the servlet mapping of the resource
#      covered by the security constraint.
#
#      There is no default.
#
#--------------------------------------------------------------------#
#
#      deployedwebapp.<webapp-name>.autoreloadinterval=<millisecs>
#
#      This property is used to specify whether or not a web
#      application is to be reloaded if changes are detected in the
#      implementation file. The property value is the number of
#      milliseconds between checks for changes by the Application
#      Server.
#      Reloading is not recommended for production environments.
#
#      To disable reloading, either don't specify the property
#      or specify an interval value of 0.
#
#      The default is no reloading.
#
#--------------------------------------------------------------------#
#
# ================================================================== #
#
#   Web Application Properties
#
#     These properties identify the characteristics of the components
#     that comprise the web application. These properties can be
#     split into two groups.
#
#      - Web application characteristics
#        These are the base characteristics of the web application.
#
#      - Servlet definitions
#        Defines any additional servlet characteristics within the
#        web application.
#
# ================================================================== #
#
#   Web Application Characteristics
#
#     These properties have the following syntax:
```

```
#
#      webapp.<webapp-name>.<property>=<value>
#
#      where <webapp-name> is the name of the web application
#            <property> is the property name
#            <value> is the value for the property
#
#      The web application properties are:
#
#
#---------------------------------------------------------------------#
#
#      webapp.<webapp-name>.description=<string>
#
#      This is a text description of the web application used in
#      displays and messages to help identify the web application.
#
#      The default is "Web Application: <webapp-name>".
#
#---------------------------------------------------------------------#
#
#      webapp.<webapp-name>.servletmapping=<URI-pattern>
#
#         The value of this property is a string that specifies a
#         URI-pattern that, within this web application root URI,
#         resolves to a class file that contains a servlet.
#
#         This property can be specified multiple times within a
#         web application to define multiple servlet mappings.
#
#         Ex. webapp.default_app.servletmapping=/servlet/*
#
#         If this property is not specified, the serving of requests
#         that attempt to access specific class file names will not
#         be honored within this web application unless handled by
#         an explicitly defined servlet.
#
#         There is no default.
#
#---------------------------------------------------------------------#
#
#      webapp.<webapp-name>.jspmapping=<URI-pattern>
#
#         The value of this property is a string that specifies a
#         URI-pattern that, within the web application, resolves to
#         a file that contains jsp or jhtml.
#
#         This property can be specified multiple times within a
#         web application to define multiple jsp mappings.
#
#         Ex. webapp.default_app.jspmapping=*.jsp
#         Ex. webapp.default_app.jspmapping=*.jhtml
#
#         If this property is not specified, jsp and/or jhtml
#         requests will not be honored within this web application
#         unless handled by an explicitly defined servlet.
#
#         There is no default.
#
#---------------------------------------------------------------------#
#
#      webapp.<webapp-name>.jsplevel=<JSP-spec-level>
#
#         This property defines the level of the JSP processor to
#         be configured for this application. This property is
#         ignored if property
#         webapp.<webapp-name>.jspmapping=<URI-pattern>
#         is not defined within the web application.
#
#         <JSP-spec-level> is either "1.1" to configure the JSP processor
#            that supports the JSP 1.1 Specification Level, "1.0" to configure
#            the JSP processor that supports the JSP 1.0 Specification Level;
#            otherwise, the JSP processor for ".91" is configured.
#
#
#         The default is the ".91" JSP processor.
#
#---------------------------------------------------------------------#
#
#      webapp.<webapp-name>.filemapping=<URI-pattern>
#
#         This property defines a URI-pattern that maps to static
#         content that you want to be served within this web
#         application. Static content is considered all content other
#         then servlets and jsp/jhtml.
#
#         The Application Server streams all static content without
#         performing any character conversions.
#
#         Ex. webapp.default_app.filemapping=*.gif
#
#         The default is that no static content can be served within
#         this web application.
#
#---------------------------------------------------------------------#
#
#      webapp.<webapp-name>.attributes=
#         <parm_1_name>=<parm_1_value>,<parm_2_name>=<parm_2_value>
#
#         <parm_n_name> specifies the name of a parameter
#
#         <parm_n_value> specifies the associated value which is
#                        treated as a string.
#
#         This property defines attributes for the web application.
#         For example, to specify attributes called x, y and z that
#         are to be available to servlets/jsps within the web
#         application "default_app", the following line would be
#         inserted:
#
#         webapp.default_app.attributes=x=0,y=Fred,z=true
#
#---------------------------------------------------------------------#
#
#      webapp.<webapp-name>.errorpagemapping=<URI-pattern>
#
#         This property defines a URI-pattern that will map to a
#         servlet or jsp that is written to handle error reporting
#         for exceptions thrown by servlets within the web application.
#
#         IBM provides a default error reporter.
#
#---------------------------------------------------------------------#
#
#      webapp.<webapp-name>.filter.<MIME-type>=<servlet-name>
#
#         <MIME-type> is a file type, such as text/html,  recognized
#         by the virtual host in which the web application is
#         deployed.
#
#         <servlet-name> is the servlet to be invoked when the
#         associated MIME type is recognized.
#
# =================================================================== #
```

```
#
#   Servlet Definitions
#
#     These properties have the following syntax:
#
#      webapp.<webapp-name>.servlet.<servlet-name>.<property>=<value>
#
#      where <webapp-name> is the name of the web application
#            <servlet-name> is the name of the servlet
#            <property> is the property name
#            <value> is the value for the property
#
#      The servlet properties are:
#
#--------------------------------------------------------------------#
#
#      webapp.<webapp-name>.servlet.<servlet-name>.servletmapping=
#                                  <URI-pattern>
#
#         This should be on one line; it is split here because of
#         spacing constraints.
#
#         <URI-pattern> is the path for the servlet, relative
#         to the web application's root URI.
#         Use a wild-card character (*) only at the beginning
#         of a path.
#
#         <servlet-nane> is the servlet being invoked.
#         For example, after you set up a servletmapping, you
#         can connect to the servlet by entering the URI-pattern
#         into a URL following the web application's root URI.
#
#         For example, to create a servletmapping for the Big servlet
#         in web application default_app, which has a root URI of
#         "/Default" that allows it to be invoked at the browser
#         by the string /Default/servlet/myCompany/Big,
#         the following line would be inserted:
#
#         webapp.default_app.servlet.Big.servletmapping=
#                                         /servlet/myCompany/Big
#
#--------------------------------------------------------------------#
#
#      webapp.<webapp-name>.servlet.<servlet-name>.code=<servlet-class>
#
#         <servlet-name> is the unique name of the servlet.
#         The name should not include double-byte characters.
#
#         <servlet-class> is the associated class file for the
#         servlet.
#
#         You do not need to specify this property if the servlet
#         name and class name are the same.
#
#         For example, to add a servlet named Big in web application
#         default_app that was compiled in file BigServlet.class
#         and is part of package com.abc,
#         the following line would be inserted:
#
#         webapp.default_app.servlet.Big.code=com.abc.BigServlet
#
#--------------------------------------------------------------------#
#
#      webapp.<webapp-name>.servlet.<servlet-name>.initargs=
#         <parm_1_name>=<parm_1_value>,<parm_2_name>=<parm_2_value>
#
#         Above should be on one line but split for spacing.
```

```
#
#         <parm_n_name> specifies the name of a parameter
#
#         <parm_n_value> specifies the associated value which is
#                        treated as a string
#
#         For example, to specify parameters called x, y, and z that
#         are to be passed to the init method for servlet Big within
#         web application default_app,
#         the following line would be inserted:
#
#         webapp.default_app.servlet.Big.initArgs=x=0,y=Fred,z=true
#
#-------------------------------------------------------------------#
#
#      webapp.<webapp-name>.servlet.<servlet-name>.autostart=true|false
#
#        Property indicates whether the servlet should be loaded; and
#        its init method driven, whenever the Application Server starts.
#
#        Default is not to autostart a servlet.
#
# ================================================================== #
#
#   Migrating a version 1.x was.conf properties file to
#   version 3.50
#
# ================================================================== #
#
#  The following are the minimal set of properties required to
#  configure an AppServer V3.50 server to support a migration from
#  AppServer v1.x. You should be able to update the properties, where
#  required, with environment-specific data and then uncomment the
#  properties and start the Application Server using these settings.
#
#  - Server properties
#
#    libpath - Propagate any libraries you added to the version 1.x
#          was.conf ncf.jvm.libpath over to the version 3.50
#          appserver.libpath.  Don't propagate any of the libraries
#          that were on the default version 1.2 was.conf. That is,
#          you should not include libraries required by the
#          application server itself. The 3.50 version of the
#          Application Server will automatically add the libraries
#          it requires to the libpath. Again, only specify libraries
#          which you have added in support of your applications.
#
#    classpath - As with the libpath, you should propagate only the
#          files (jars, zips, .ser) you added to the version 1.x
#          was.conf ncf.jvm.classpath property. Do not propagate any
#          of the files that resided on the default ncf.jvm.classpath.
#          The version 3.50 Application Server will automatically
#          add files it requires to the classpath.
#
#          There is a further discussion on classpath consideration
#          in the section regarding web application classpath.
#
#  - Session properties
#
#    - The session properties supported in the version 1.x was.conf
#      correspond one-to-one with properties in the version 3.50
#      was.conf. To start the server with equivalent session support
#      configured, you need to copy the directives prefixed with
#      session.* from your existing WAS.conf file to this file.
#
#      One point to note is that the session support in version 3.50
#      defaults to enable=true.  This can be overridden by the
#       session.enable property.
#
#   - Logging properties
#
#       The logging properties have changed between version 1.x and
#       version 3.50. It is recommended that you start with the default
#       logging properties of version 3.50 and modify as needed.
#
#   - Virtual Host
#     - Define a host called "default_host".
#     - Take the default mime types.
#     - You must replace <your-hostname> with your specific hostname
#       (for example host.default_host.alias=www.mycompany.com:8027).
#
#       Note: You can have multiple alias statements for a single
#       host. If you want more than one DNS alias to map to a host,
#       just add multiple configuration directives.
#
#####   host.default_host.alias=<your-hostname>
#
#   - Web Application
#     - Define a web application called "default_app".
#     - Deploy default_app into the virtual host default_host.
#
#####   deployedwebapp.default_app.host=default_host
#
#     - Establish a URI namespace within the application of "/" for
#       default_app. This provides all content deployed within the
#       default_app with the view that their namespace is rooted
#       at the base of the virtual host.
#
#####   deployedwebapp.default_app.rooturi=/
#
#     - Establish a document root for JSP, JHTML and static content
#       served within the Web Application. You must replace
#       <your-document-root> with the directory that contains your
#       JSP and JHTML.
#
#
#####   deployedwebapp.default_app.documentroot=<your-document-root>
#
#     - Establishes the classpath for the default_app web application;
#       this classpath can be reloadable and is searched after the
#       JVM classpath.  In version 1.x of the Application Server,
#       the server would always search the JVM classpath, followed
#       by the /servlets directory in the server_model_root, followed
#       by the reloadable classpath specified by the
#       servlet.reload.directories.  To maintain this behavior in
#       the v3.50 Application Server, you need to consider
#       the following:
#
#       - JVM classpath - The version 3.50 Application Server
#           automatically constructs the classpath with the
#           jar and zips that are required to operate. This includes
#           Application specific libraries, as well as any required
#           JDK libraries. Therefore, you should propagate only the
#           libraries you added to the version 1.x ncf.jvm.classpath
#           onto the version 3.50 appserver.classpath.
#       - Reloadable classpath - The version 3.50 Application Server
#           supports a web application classpath which is searched after
#           the JVM classpath in the order in which the libraries
#           appear in the classpath. This classpath can be configured
#           to be reloadable. To maintain a consistent search
#           order with the version 1.x Application Server, you
#           should add your <server-model-root>/servlets directory
#           followed by the reloadable directories.
#
```

```
#          Note: The verion 3.50 Application Server has no
#            requirement to be pointed at the same instance
#            of the /servlets directory used in your version 1.x
#            Application Server. You may in fact choose to make
#            a copy of that directory and its subdirectories
#            before using it within a version 3.50 Application
#            Server in anticipation of the possible need to
#            migrate either servlets or JSP to the new APIs
#            APIs supported in version 3.50.
#
#####  appserver.classpath=<your-libraries-for-the-jvm-classpath>
#####  deployedwebapp.default_app.classpath=<your-reloadable-classpath>
#
#    - Servlet reload - The property to control servlet reloading in
#      version 1.x was.conf was servlets.reload.  In the version 3.50
#      was.conf, servlet reloading is controled, per web application,
#      via the following property.  The value of the property is the
#      interval, in milliseconds, that the Application Server should
#      pool for changes. To disable reloading, either don't specify
#      the property or specify an interval value of 0.
#
#####  deployedwebapp.default_app.autoreloadinterval=<milliseconds>
#
#    - Enable servlet requests to be resolved to a file name.  This
#      property corresponds to the urltype.servlets= property in
#      the version 1.x was.conf file. For each instance of the
#      property defined in the version 1.x was.conf, add an
#      instance of the following property with the corresponding
#      value. For example, the default was.conf for version 1.x
#      defined urltype.servlet=/servlet. This same behavior is
#      represented in version 3.50 with the following property
#      within the default_app web application.
#
#####  webapp.default_app.servletmapping=/servlet/*
#
#    - Enable jsp and jhtml requests to be processed. This property
#      corresponds to the urltype.jsp= property in the version 1.x
#      was.conf.
#
#####  webapp.default_app.jspmapping=*.jsp
#####  webapp.default_app.jspmapping=*.jhtml
#
#    - Filtering by mime-type
#
#      version 1.x property
#          filter.<mime-type>=<servlet-1>,<servlet-2>,...
#      version 3.50 property
#         Version 3.50 Application Server supports one servlet
#         name per mime-type.
#
#####    webapp.default_app.filter.<MIME-type>=<servlet-name>
#
#    - Servlet properties have a direct mapping between version 1.x and
#      version 3.50.
#
#      code -
#        version 1.x property
#          servlet.<servlet-name>.code=<servlet-class>
#        version 3.50 property
#
#      initargs -
#        version 1.x property
#          servlet.<servlet-name>.initArgs=<initargs>
#        version 3.50 property
#
#      autostart -
#        version 1.x property
#          servlets.startup=<servlet-name1> <servlet-name2> ...
#        version 3.50 property
#          Each unique servlet you want to have started when the
#          Application Server starts requires an autostart property.
#          Do not define an autostart property for the invoker servlet
#          from the default startup property in the version 1.x
#          was.conf.
#
#      alias -
#        version 1.x property
#          servlet.<alias-name>=<servlet-name>
#        version 3.50 property is servlet mapping
#
#####      webapp.default_app.servlet.<servlet-name>.servletmapping=<alias-name>
#####      webapp.default_app.servlet.<servlet-name>.autostart=true
#####      webapp.servlet.<servlet-name>.code=<servlet-class>
#####      webapp.default_app.servlet.<servlet-name>.initargs=<initargs>
#
################################################################################
```

Appendix C. default_global.properties file

Following is a copy of the default JVM properties file that is provided with the Application Server. This file is located in the *applicationserver_root*/**properties/default_global.properties** file.

```
#################################################################
# @(#)default_global.properties 1.0 98/08/21
#
# Configuration properties for JVM and plugin initialization
#
#################################################################


#################################################################
#
#  Customer modifiable jvm config options
#
#################################################################

#################################################################
#
#  JVM Configuration jit setting
#
#  Turn jit on or off, or specify a different jit compiler.
#
#  Default:  on
#
#  Syntax:   appserver.product.java.jvmconfig.jit=on | off | jitc
#
#  Example:  appserver.product.java.jvmconfig.jit=on
#
#################################################################
# appserver.product.java.jvmconfig.jit=

#################################################################
#
#  JVM Configuration maximum heap size setting
#
#  Default:  128m
#
#  Syntax:   appserver.product.java.jvmconfig.mx=maxmem[k | m]
#
#  example:  appserver.product.java.jvmconfig.mx=64m
#
#################################################################
# appserver.product.java.jvmconfig.mx=

#################################################################
#
#  JVM Configuration initial heap size setting
#
#  Default:  128m
#
#  Syntax:   appserver.product.java.jvmconfig.ms=initmem[k | m]
#
#  Example:  appserver.product.java.jvmconfig.ms=64m
#
#################################################################
# appserver.product.java.jvmconfig.ms=

#################################################################
#
```

```
#  JVM Configuration Java stacksize setting
#
#  Default:  400k
#
#  Syntax:   appserver.product.java.jvmconfig.oss=stacksize[k | m]
#
#  Example:  appserver.product.java.jvmconfig.oss=500k
#
################################################################
# appserver.product.java.jvmconfig.oss=

################################################################
#
#  JVM Configuration native stacksize setting
#
#  Default:  256k
#
#  Syntax:   appserver.product.java.jvmconfig.ss=stacksize[k | m]
#
#  Example:  appserver.product.java.jvmconfig.ss=512k
#
################################################################
# appserver.product.java.jvmconfig.ss=


################################################################
#
#  Application Server run byte-code verifier
#
#  Default:  false
#
#  Syntax:   appserver.product.java.jvmdebug.verify=true | false
#
#  Example:  appserver.product.java.jvmdebug.verify=true
#
################################################################
# appserver.product.java.jvmdebug.verify=

################################################################
#
#  Application Server use Java debug library
#
#  Default:  false
#
#  Syntax:   appserver.product.java.jvmdebug.debug=true | false
#
#  Example:  appserver.product.java.jvmdebug.debug=true
#
################################################################
# appserver.product.java.jvmdebug.debug=

################################################################
#
#  JVM remote debug port. This property is used only when
#  appserver.product.java.jvmdebug.debug=true.
#
#  Default:  None
#
#  Syntax:   appserver.product.java.jvmdebug.port=<port_number>
#
#  Example:  appserver.product.java.jvmdebug.port=8888
#
################################################################
# appserver.product.java.jvmdebug.port=

################################################################
#
```

```
#  Application Server print message when garbage collection frees
#  memory
#
#  Default:  false
#
#  Syntax:   appserver.product.java.jvmdebug.verbosegc=true | false
#
#  Example:  appserver.product.java.jvmdebug.verbosegc=true
#
################################################################
# appserver.product.java.jvmdebug.verbosegc=

################################################################
#
#  Application Server print message when classes load
#
#  Default:  false
#
#  Syntax:   appserver.product.java.jvmdebug.verbose=true | false
#
#  Example:  appserver.product.java.jvmdebug.verbose=true
#
################################################################
# appserver.product.java.jvmdebug.verbose=
#
#
#  This line added as an aid to users of OEDIT that want to create long
#  lines. It can be deleted with no effect on the Application Server.
####################################################################
```

Appendix D. Programming Model Restrictions

As with Version 3.02, it is an objective of WebSphere Application Server Standard Edition Version 3.5 for OS/390 to fully support the same interfaces and facilities that are supported on WebSphere Application Server Standard Edition Version 3.5 for MultiPlatforms. Therefore, applications developed on WebSphere Application Server Standard Edition Version 3.5 for MultiPlatforms can be re-deployed unchanged to WebSphere Application Server Standard Edition Version 3.5 for OS/390.

There are known Programming Model exceptions for this version of the Application Server for OS/390. It does not currently provide support for:

- The User Profile APIs that are present on Version 3.5 Standard and Advanced Edition Application Servers for MultiPlatforms. This support will be added to the product in a future service update.
- The Java Transaction API (JTA) interfaces. Servlets executing on OS/390 cannot explicitly begin and commit transactions using the JTA APIs. It is possible to execute requests to databases such as DB2 using JDBC and existing transactional systems, such as IMS, using Common Connector Framework support.

  The ability to commit and roll back requests made to these systems is provided within the driver implementations for accessing these systems. You can perform a commit and roll back of DB2 resources using the SQL Commit capability provided by the JDBC driver. However, you cannot begin a transaction using the JTA interfaces, drive requests to DB2 over JDBC, and then subsequently commit or roll back these changes using the JTA interfaces. This capability will be provided in a future version of the product.
- The ″session-timeout″ XML tag in .webapp configuration files is not supported. You can configure the maximum inactive time for HttpSessions for an instance of the Application Server by setting the ″session.invalidationtime″ value in the was.conf file.

This version of the Application Server for OS/390 exists within the IBM HTTP Server and is able to receive HTTP requests. This enables this version of the Application Server to exist in DMZ configurations, including routers, proxy servers, etc., that send HTTP requests to the originating Web servers.

This version of the Application Server for OS/390:

- Does not provide support for single sign-on capability with Application Servers executing on distributed platforms. Instead, it makes use of the security facilities provided by OS/390.
- Cannot participate in a WebSphere redirector configuration which routes requests via IIOP or OSE from an intermediary Web server (often referred to as a redirector) to a Application Server host containing the servlets and JSPs.

These capabilities will be added in a future version of the product.

Appendix E. Enabling subsystems for use with the Application Server

This appendix discusses the customization needed to enable communication between the Application Server and the following OS/390 subsystems:

- DB2
- CICS
- IMS

Enabling communication with DB2

Before the Web servers and their Java applications can access DB2, JDBC and/or SQLJ support must be available on the OS/390 system on which they are running.

- JDBC is an application programming interface (API) that Java applications can use to access any relational database.
- SQLJ is an API that provides support for embedded static SQL in Java applications.

Because DB2 for OS/390 SQLJ support includes JDBC, SQLJ applications can also execute dynamic SQL statements through JDBC. To use DB2 for OS/390, you must:

- Install one of the following DB2 products:
  - DB2 for OS/390 Version 5
  - DB2 UDB for OS/390 Version 6
  - DB2 UDB for OS/390 and z/OS Version 7

  Note: Hereafter, all of these versions will simply be referred to as DB2.
- Install the DB2 JDBC driver.
- Install Version 3.5 of the Application Server. (See Chapter 2, “Installing and customizing the Application Server”, on page 2-1 for installation instructions.)
- Update the Application Server and Web server configuration files to enable the Application Server to locate, and communicate with, DB2.
- Set up DB2 tables to contain the data the servlets require.
- Configure the Recoverable Resource Manager Services Attachment Facility (RRSAF), if it is not already configured, and enable multicontext support:
  1. Established a profile for controlling access from RRS by defining ssnm.RRSAF in the DSNR resource class with a universal access authority of NONE.
  2. Activate the resource class:
     ```
     SETROPTS RACLIST(DSNR) REFRESH
     ```

3. Add user IDs that are associated with the stored procedures address spaces to the RACF Started Procedures Table:

   ```
   DC CL8'DSNWLM' WLM-ESTABLISHED S.P. ADDRESS SPACE
   DC CL8'SYSDSP' USERID
   DC CL8'DB2SYS' GROUP NAME
   DC X'00' NO PRIVILEGED ATTRIBUTE
   DC XL7'00' RESERVED BYTES
   ```
4. Allow read access to ssnm.RRSAF to the user ID associated with the stored procedures address space.

- Start the Web server.

When the Web server starts, the changes made in the Application Server and Web server configuration files will become effective, and DB2 and the Application Server should be able to communicate with each other. Invoke a servlet that uses the DB2 database at this time to ensure that a proper interface has been established.

Installing DB2

If DB2 for OS/390 is already installed on your OS/390 system, and you are using either Version 5 or Version 6, make sure one of the following PTFs is applied:

- PTF UQ49041, if you are using DB2 Version 5.
- PTF UQ49039, if you are using DB2 Version 6.

If DB2 for OS/390 is not already installed on your OS/390 system:

- Install DB2. See one of the following pairs of publications for a description of how to install a specific version of DB2:
  – *DB2 for OS/390 Program directory*, GI10-6973-05, and *DB2 for OS/390 V5 Installation Guide*, GC26-8970
  – *IBM Database 2 Universal Database Server for OS/390 Program Directory*, GI10-8182-04, and *DB2 UDB for OS/390 V6 Installation Guide*, GC26-9008
  – *DB2 Universal Database for OS/390 and z/OS V7 Program Directory*, GI10-8216-01, and *DB2 UDB for OS/390 and z/OS V7 Installation Guide*, GC26-9936
- Apply PTF UQ49041 (for DB2 Version 5) or PTF UQ49039 (for DB2 Version 6).

The DB2 program directories are shipped with the respective products. All of the other documents can be accessed by entering the following URL from your browser and then selecting the appropriate version of DB2:

```
http://www.ibm.com/software/data/db2/os390/library.html
```

Installing a JDBC driver

If a DB2 JDBC driver is already installed on your OS/390 system, see “Enabling the Application Server to locate, and communicate with, DB2” on page E-3 for a description of what needs to be done to ensure that the Application Server knows where to find it.

If a DB2 JDBC driver is not already installed on your OS/390 system, the following publications describe how to install a JDBC driver for DB2:

- *DB2 for OS/390 V5 Application Programming Guide and Reference for Java*™, SC26-9547
- *DB2 UDB for OS/390 V6 Application Programming Guide and Reference for Java*™, SC26-9018

- *DB2 UDB for OS/390 and z/OS V7 Application Programming Guide and Reference for Java™*, SC26-9932

To access these documents, enter the following URL from your browser and then select the version of DB2 that you will be using with the JDBC driver:

`http://www.ibm.com/software/data/db2/os390/library.html`

When JDBC installs, it creates a Samples subdirectory on the JDBC driver which contains JDBC samples. Have your DB2 Administrator update these samples with your system-specific information, such as the location of DB2, the correct URL, and so forth, and have him give you authority to use them. Then run one of the samples before starting the Web server.

- If the sample runs successfully and JDBC can access the DB2 data, this indicates you have successfully installed and configured the DB2 JDBC driver.
- If the sample does not run successfully, there is probably a problem with your JDBC installation that your DB2 Administrator needs to fix before the Web server is started.

For more information about updating and running these samples and how to use DB2 for OS/390 JDBC when writing Java applications, see one of the following publicaitons:

- *DB2 for OS/390 V5 Application Programming Guide and Reference for Java™*, SC26-9547
- *DB2 UDB for OS/390 V6 Application Programming Guide and Reference for Java™*, SC26-9018
- *DB2 UDB for OS/390 and z/OS V7 Application Programming Guide and Reference for Java™*, SC26-9932

Enabling the Application Server to locate, and communicate with, DB2

Before DB2 can be used to store data required by servlets running under the Application Server, the Application Server must be able to locate, and communicate with, DB2,. To set up communication between DB2 and the Application Server you must:

1. Make the following changes to the Application Server was.conf file:
   - Add the name of the DB2 JDBC driver classes file to the **appserver.classpath** property.

     If you are using DB2 Version 5 or DB2 Version 6, **db2jdbcclasses.zip** is the name of the JDBC driver classes file.

     If you are using DB2 Version 7, the name you specify on the **appserver.classpath** property depends on which JDBC driver your DB2 administrator has selected to use:

     – If the driver that is based on the JDBC 1.2 specification has been selected, specify any one of the following files:
       - **db2jdbcclasses.zip**
       - **db2sqljclasses.zip**
       - **db2sqljruntime.zip**

     – If the driver that is based on the JDBC 2.0 specification has been selected, specify **db2j2classes.zip**.

Adding the name of the DB2 JDBC driver classes file to this property enables the Application Server to locate the DB2 JDBC driver. If DB2 was installed using the instructions in the DB2 Program Directory, and the DB2 Installation Guide, all of these file will be located in the **/usr/lpp/db2/db2*nn*0/classes** directory. (*nn* is 51 for DB2 V5, 61 for DB2 V6, or 71 for DB2 V7.)

- Add the directory containing the **DSNJDBC_JDBCProfile.ser** file to the **appserver.classpath** property after the name of the DB2 JDBC driver classes file. (The **DSNJDBC_JDBCProfile.ser** file is created when the db2genJDBC program is run during the DB2 JDBC driver installation.) If DB2 is installed using the instructions in the DB2 Program Directory, and the DB2 Installation Guide, this file will be located in the **/usr/lpp/db2/db2*nn*0/classes** directory. (*nn* is 51 for DB2 V5, 61 for DB2 V6, or 71 for DB2 V7.)

  If the DB2 JDBC driver was installed into the default directories, this file will be located in the **/usr/lpp/db2/db2*nn*0**, directory. (*nn* is 51 for DB2 V5, 61 for DB2 V6, or 71 for DB2 V7.)
- Add the location of the DB2 JDBC driver to the values specified on the **appserver.libpath** property. If DB2 was installed using the instructions in the DB2 Program Directory, and the DB2 Installation Guide, this file will be located in the **usr/lpp/db2/db2*nn*0/lib** directory. (*nn* is 51 for DB2 V5, 61 for DB2 V6, or 71 for DB2 V7.)

  Note: If you prefer, you can obtain the same results by adding the following values to the LIBPATH variables in the Web server httpd.envvars configuration file:

  ```
  /usr/lpp/db2/db2nn0/lib
  ```

A template of the was.conf file is located in the *applicationserver_root***/properties** directory. See “Customizing the Application Server” on page 2-8 for more information about updating this file.

Note: If you prefer, you can place these classpath entries in your HTTP Server’s httpd.envvars file and set the **appserver.usesystemclasspath** was.conf file property to true. Directories mentioned in the ″system classpath″ are added to the end of the generated classpath without undergoing the ″archive expansion″ process.

2. Add the following environment variables to the end of the Web serve’sr httpd.envvars configuration file:

   ```
   DB2SQLJPROPERTIES=/usr/lpp/db2/db2nn0/classes/db2sqljjdbc.properties
   ```

   nn is 51 for DB2 V5, 61 for DB2 V6, or 71 for DB2 V7.

   Note: A dsnaoini file is not required for a DB2 JDBC driver at this service level.
3. Add the following DD statements to the Web server’s STEPLIB DD statement:

   ```
   //         DD  DSN=DSNnn0.SDSNEXIT,DISP=SHR
   //         DD  DSN=DSNnn0.SDSNLOAD,DISP=SHR
   ```

   If you are using DB2 V7, you must also add the the following DD statement in the STEPLIB statement in order to include the SDSNLOD2 library:

   ```
   //         DD  DSN=DSNnn0.SDSNLOD2,DISP=SHR
   ```

   Remember that the DSN= portion of these statements must start no later than column 16.

Setting up DB2 tables

Even if the Application Server is able to locate, and communicate with, DB2, the DB2 Administrator must set up DB2 tables to contain the data the servlets require before data can be passed between DB2 and the Application Server. You should work with your servlet developers and DB2 Administrator to set up the required DB2 tables. See one of the following publications for a description of the syntax of the SQL statements used to define DB2 tables:

- *DB2 for OS/390 V5 SQL Reference*, SC26-8966
- *DB2 UDB for OS/390 V6 SQL Reference*, SC26-9014
- *DB2 UDB for OS/390 and z/OS V7 SQL Reference*, SC26-9944

Customizing SQLJ/JDBC run-time properties files

The DB2 JDBC driver's default settings for the DB2 attach type and OS/390 multiple context support are correct for Application Server use. However, you may want to refer to the *DB2 Application Programming Guide and Reference for Java* for the version of DB2 you are using for information about the other JDBC driver properties that can be set within this file.

Enabling communication with CICS

Before the Application Server can communicate with CICS, the CICS Transaction Server must be installed and working. *CICS Transaction Gateway: OS/390 Gateway Administration*, SC34-5935, describes how to install the CICS Transaction Gateway for OS/390 and how to configure the CICS Transaction Server.

The EXCI must also be correctly configured. See *CICS External Interfaces Guide*, SC33-1944 for details.

Information about how CICS can be used to exploit Java based technologies can be found at URL:

```
http://www.ibm.com/software/ts/cics/about/modern/cicsjava.html
```

Preparing the Application Server for CICS TS

The following changes must be made to the Application Server, the Web server, and CICS before Java servlets can communicate with the CICS Transaction Server:

1. Add the following jar files to the **appserver.classpath** property in the Application Server was.conf file:

   ```
   /usr/lpp/cicsts/ctg/classes/ctgclient.jar
   /usr/lpp/cicsts/ctg/classes/ctgserver.jar
   ```

   It is recommended that these files be placed near the beginning of the list of files/directories specified on this property.
2. Add the following directory to the **appserver.libpath** property in the Application Server was.conf file:

   ```
   /usr/lpp/cicsts/ctg/bin
   ```

   It is recommended that this directory be placed near the beginning of the list of files/directories specified on this property.
3. Add the CICS Transaction Server SDFHEXCI library as a STEPLIB to your Web server startup procedure.
4. Make sure the following program control has been added:

```
RALTER PROGRAM * ADDMEM ('CICSTS13.CICS.SDFHEXCI&#146;//NOPADCHK) UACC(READ)
SETR WHEN(PROGRAM) REFRESH
SETR  RACLIST(FACILITY) REFRESH
```

Make sure the extended program attributes have also been turned on. You can check this by issuing the following command:

```
ls -E /ctg/bin/lib*.so
```

If required, the following command will turn on extended program attributes:

```
extattr +p /ctg/bin/lib/*.so
extattr +p /ctg/lib/*.SECURES
```

5. Make sure the DFH$EXCI group has been installed. This group defines session and connection resources to CICS.
6. If you are not running CTG in local mode, you must specify that ctgstart script does not share its address space with any other processes. To force the JVM to use its own non-sharable address space, enter:

   ```
   extattr -s /ctg/bin/ctgstart
   ```

   If you are concerned about performance, it is recommended that you run CTG in local mode. In this case, you must set:

   ```
   JAVA_PROPAGATE=NO
   ```

For more information about CICS configuration settings, see *CICS Transaction Gateway V4.0 OS/390 Gateway Administration*, SC34-5935, and the *Information Center v1.0 for CICS Transaction Server v2.1*, at URL:

`http://www-4.ibm.com/software/ts/cics/library/infocenter/`

Enabling communication with IMS using IMS Connect and IMS Connector for Java

IMS Connect replaces IMS TCP/IP OTMA Connection (ITOC). IMS Connect is an SMP/E installable and maintainable product, and provides several new functions and enhancements not available with ITOC. ITOC should only be used for ″proof of concept″ purposes, since support for ITOC will be withdrawn March 1, 2001. IMS Connect allows TCP/IP clients, IMS Connector for Java in particular, to send messages to the IMS Transaction Manager through the IMS Open Transaction Manager Access (OTMA) interface.

IMS Connect 1.1 and IMS Connector for Java 3.5.3 includes Local Option support. Local Option enables non-TCP/IP communication between IMS Connector for Java and IMS Connect when IMS Connector for Java is used with WebSphere Application Server for OS/390. In addition, both the IMS Connector for Java and IMS Connect address spaces must reside in the same MVS image. Local Option uses MVS Program Calls to communicate between the Application Server and IMS Connect address spaces.

Local Option is supported only with IMS Version 7.1. APAR PQ45057 must be applied to IMS Connect in order to enable the Local Option support in IMS Connect that allows IMS Connector for Java to use Local Option.

Local Option is not supported with IMS Version 5.1 or IMS Version 6.1. However, IMS Connector for Java 3.5.3 can be used with IMS Connect 1.1 and IMS Version 5.1 or 6.1. For additional information on IMS Connect, including access by clients other than IMS Connector for Java, see:

`http://www.ibm.com/software/data/ims/imstoc.html`

IMS Connector for Java

IMS Connector for Java, one of the e-business connectors included in VisualAge for Java, can be used by servlets to access IMS transactions. IMS Connector for Java is used, in conjunction with VisualAge for Java's Enterprise Access Builder (EAB) tools, to create an EAB command that accesses the IMS transaction. This EAB command is typically provided to IBM's WebSphere Studio to generate a servlet.

To run servlets developed with VisualAge for Java Enterprise Edition V 3.5.3, you must:

- Have the IMS Connect product installed and running.
- Make the following class libraries accessible to the Application Server:
  - The IMS Connector for Java, found in *<vajava_install_drive>*/**IBM Connector/classes/imsconn.jar**.
  - The J2EE Connector Architecture, found in *<vajava_install_drive>*/**IBM Connector/classes/connector.jar**.
  - The IBM Enterprise Access Builder Library, found in *<vajava_install_dir>*/**eab/runtime30/eablib.jar** .
  - The IBM Java Record Library, found in *<vajava_install_dir>*/**eab/runtime30/\recjava.jar**.
  - The Local Option Shared Library, found in *<vajava_install_drive>*/**IBM Connector/lib/libimsconn.so** .

<vajava_install_drive> is the drive on which VisualAge for Java is installed; for example, d:, and *<vajava_install_dir* is the directory where VisualAge for Java is installed; for example, d:\VAJava.

Note: The Local Option Shared Library is only needed if you are using Local Option. Also, VisualAge for Java's Common Connector Framework (ccf.jar) file is not copied, since the Application Server provides its own implementation of ccf.jar. For this implementation, the RuntimeContext object that is associated, by default, with the servlet's thread of execution has a global (common) ConnectionManager that supports connection pooling.

If you are running servlets that inherit from Studio class StudioPervasiveServlet of package com.ibm.webtools.runtime, ensure that the class StudioPervasiveServlet is deployed in your Application Server environment. You can do this by using VisualAge for Java to create a JAR file of the package com.ibm.webtools.runtime (for example, webtools.jar), and then deploy it to your Application Server environment.

The first four files in the above list, and, for example, webtools.jar if needed, must be uploaded to the Host, stored in your HFS, and added to the appserver.classpath property in the Application Server was.conf file. The file libimsconn.so, if used, needs to be uploaded to the Host, stored in your HFS, and added to the appserver.libpath property in the Application Server was.conf file.

The JAR files containing the Enterprise Access Builder (EAB) commands that the servlets running on the WebSphere Application Server for OS/390 execute also need to be added to the appserver.classpath property in the Application Server was.conf file.

Appendix F. Using the Connection Manager APIs

Connection Manager support will be deprecated in future releases of the Application Server. Therefore you should not use the Connection Manager for new application development.

If you have applications that currently use the Connection Manager, the information contained in this appendix may be useful.

How a servlet uses the Connection Manager

Any servlet using the Connection Manager must follow the following steps:

1. Create the connection specification:

   The servlet prepares a specification object identifying information necessary for connection to a DB2 JDBC database.

   If the connection pool property, connectionidentity=server, is specified for the Connection Manager pool, the Application Server gets a connection using the Web server's identity, even if %%CLIENT%% is specified or a userid and password are coded in the connection specification. If connectionidentity=connspec, the Application Server gets a connection using the identity specified by the user and password in the connection specification.

   If the connection pool property, connectionidentity=thread, is specified for the Connection Manager pool, the Application Server gets a connection using the identity of the thread on which the connection request is made. With %%CLIENT%% this will be the userid entered on the browser. With %%SERVER%% this will be the identity of the Web server address space. See the notes on the connectionidentity property for a Connection Manager pool under IBMJdbcConnSpec class for more information.

2. Connect to the Connection Manager:

   The servlet gets a reference to the Connection Manager in order to communicate with it. This needs to be done only once in a servlet's lifetime.

3. Get a Connection Manager connection:

   The servlet asks the Connection Manager for a connection to a DB2 JDBC database using the connection specification prepared in step 1. The connection object returned is from a Connection Manager pool, and is an instance of a class defined in the Connection Manager APIs – it is not an object from a class in the JDBC API set. This first connection is called a Connection Manager connection. Usually, a servlet gets a Connection Manager connection for every user request.

4. Use the Connection Manager connection to access a pre-established DB2 connection:

   The servlet invokes a method on the Connection Manager connection returned in step 3, retrieving an object defined in the JDBC API set. This object is called a DB2 connection to distinguish it from a Connection Manager connection.

   The DB2 connection, unlike the Connection Manager connection, is from the underlying JDBC API set. The DB2 connection is not *created* for the servlet – the servlet instead uses the pre-established DB2 connection by virtue of owning a Connection Manager connection from the pool.

The DB2 connection will be used for the actual interactions with DB2, using the methods from the underlying JDBC API set. The JDBC APIs are found in the java.sql package.

5. Interact with DB2:

   The servlet interacts with DB2. It retrieves data, updates data, etc., using JDBC methods. (Information about these methods can be found in the documentation for the java.sql package and in the documentation that comes with DB2.)

   Note: If a servlet uses a DB2 connection for more than one interaction within the same user request, before performing each interaction, your developers should verify that the servlet still owns the associated Connection Manager connection. The Connection Manager periodically checks a last-used timestamp to see if your servlet has been using the Connection Manager connection, and if not, the Connection Manager assumes that your servlet has failed or has otherwise become unresponsive. It takes the Connection Manager connection away. Verifying the Connection Manager connection (using the verifyIBMConnection() method) also updates the last-used timestamp.

6. Release the connection:

   The servlet returns the Connection Manager connection to the Connection Manager pool, freeing the connection for use by another servlet or by another request from the same servlet.

7. Prepare and send the response:

   The servlet prepares and returns the response to the user request. In this step the developer will probably not be using any Connection Manager APIs.

Connection Manager APIs

The following sections describe the Connection Manager APIs and illustrate how they relate to the steps discussed in “How a servlet uses the Connection Manager” on page F-1.

IBMJdbcConnSpec class

A servlet must include a JDBC specification object to record the specifications for the connection to DB2. This is typically done in step 1 on page F-1. Note that the specification object does not actually set the specifications, but is used as an argument by the method that gets the connection (see the getIBMConnection() method in the IBMConnMgr class below). After the servlet constructs the specification object, it can use get and set methods to specify the connection requirements, but usually all the requirements are included on the specification object. The constructor details are:

```
public IBMJdbcConnSpec(String poolName, boolean waitRetry, String
dbDriver, String url,String user,String password)
```

Parameters:

poolName
: The name of the Connection Manager pool containing the connection type the servlet requires.

waitRetry
: Indicates whether a servlet is to wait for a connection to free up if the pool does not currently have an available connection (set to true), or if it is to immediately fail if a connection from the pool is not available (set to false). The waitforconnectiontimeoutmilliseconds Connection Time Out parameter

for the Connection Manager pool is used to set the length of the wait for the entire connection pool. If the waitRetry parameter is set to true and a connection cannot be made is not available by the timeout within the time limit specified on the waitforconnectiontimeoutmilliseconds parameter, the connection request will fail.

dbDriver
The name of the DB2 driver providing the JDBC-ODBC bridge. See the DB2 Administrator for the driver name. See the Driver class in the java.sql package for more information.

url The URL for the DB2 database (typically in the form jdbc:db2os390:database name). See the getConnection() method in the DriverManager class in the java.sql package for more information.

user ID
The database user on whose behalf the connection is being made. See the getConnection() method in the DriverManager class in the java.sql package for more information.

password
The user password. See the getConnection() method in the DriverManager class in the java.sql package for more information.

When specifying a value for the connectionidentity property in the was.conf file, you should consider the following:

- The JDBC specification dictates that the user ID that is to be used for subsequent database access checks be explicitly provided as user ID and password parameters on the DriverManager.getConnection method.
- The Application Server always passes the user ID and password specified on the IBMJdbcConnSpec method as input to the DriverManager.getConnection method when obtaining a database connection.
- The JDBC driver implementation provided with the DB2 for the OS/390 product currently does not make use of the values passed in the user ID and password parameters of the DriverManager.getConnection method. Instead, it uses the identity that has been established on the currently executing thread as the primary authorization ID for use on subsequent database access checks via SQL. In order to accommodate this behavior, the Application Server always establishes a user identity on the execution thread prior to invoking the DriverManager.getConnection method.
- When the connectionidentity property is set to server, the identity of the process that is hosting the Application Server (i.e. the Web server's identity) is established on the execution thread prior to obtaining a DB2 JDBC connection.
- When the connectionidentity property is set to connspec, the user ID and password specified on the IBMJdbcConnSpec are used to establish the identity of the execution thread when obtaining a DB2 JDBC connection.
- When the connectionidentity property is set to thread, the user ID on the thread issuing the connection request is used to obtain the DB2 JDBC connection. With %%CLIENT%%, the userid on the thread will be the userid entered at the browser. With %%SERVER%%, the userid on the thread will be the identity of the Web server address space.
- JAVA_PROPAGATE=NO, the Web server must be defined as a RACF SURROGAT and all client userids permitted to the Web server SURROGAT.
- If the connectionidentity property is not specified, the default behavior is to use the user ID and password specified on the IBMJdbcConnSpec to establish the identity of the execution thread when obtaining a DB2 JDBC connection.

See the Javadoc for the Application Server APIs for information about the get and set methods for the IBMJdbcConnSpec class.

IBMConnMgrUtil class

Use a method of this class to get a reference to the Connection Manager. This is typically done in 2 on page F-1. You will use the reference to communicate with the Connection Manager and use its services.

Only one class or static method of this class is of interest – the public static IBMConnMgr.

Returns:

A reference to the Connection Manager.

IBMConnMgr class

The running instance of the Connection Manager is an instance of this class. A servlet will get a reference to the Connection Manager in step 2 2 on page F-1. The servlet will use this reference to communicate with the Connection Manager and use its services. A servlet never creates an instance of the Connection Manager, but instead uses a reference to the existing instance. For details, see the previous discription of the getIBMConnMgr() method of the IBMConnMgrUtil class.

Only one method of this class is of interest – the getIBMConnection() method to get a Connection Manager connection from the pool (used in step 3 on page F-1).

getIBMConnection()

```
public IBMConnection getIBMConnection(IBMConnSpec connSpec)
throws IBMConnMgrException
```

This method gets a Connection Manager connection from the pool for use by the servlet, if such a connection is available. Note, the only parameter passed is a specification object (created in step 1 on page F-1) for the connection. If a Connection Manager connection is not immediately available, and if waitRetry in the specification object is set to true, the servlet can wait for a Connection Manager connection to become available.

The length of the wait is set using the ConnectionTimeOut parameter for the Connection Manager pool. The ConnectionTimeOut parameter can also be used to disable the wait or extend the wait indefinitely. If a Connection Manager connection does not become available after the wait period, the getIBMConnection() method will throw an IBMConnMgrException exception.

If waitRetry is false, failure to get a Connection Manager connection causes the getIBMConnection() method to throw the exception right away.

Note: Note the returned IBMConnection object is a ″general″ connection object and must be cast to a connection object specific to DB2. Without the appropriate cast, the connection object cannot use its methods to get to DB2 in step 5 on page F-2.

Parameters:

connSpec
An extension of the IBMConnSpec class, containing detailed connection requirements for DB2 (created in step 1 on page F-1).

Returns:

An IBMConnection object from the Connection Manager pool.

IBMJdbcConn class

The IBMConnection object retrieved in step 3 on page F-1 needs to be cast to the IBMJdbcConn class in order for DB2 to be accessed. Otherwise, there is no access to the APIs associated with DB2. Access to the APIs of the JDBC server through a IBMJdbcConn object is typically established in step 4 on page F-1.

The IBMJdbcConn class has one method of interest:

```
public Connection getJdbcConnection()
```

Returns:

A Connection object to DB2.

The connection class is from the JDBC API and is documented with the java.sql package. Methods of the Connection class let you interact with DB2. Elsewhere in this chapter the Connection class is referred to as the DB2 connection, to distinguish it from the Connection Manager connection. Recall that the Connection Manager connection is not part of the JDBC API set.

IBMConnection class

The IBMJdbcConn class is an extension of the IBMConnection class. Thus, many of the methods in the IBMConnection class are also in instances of the IBMJdbcConn class. Two methods of the IBMConnection class (also in IBMJdbcConn) are of interest – the verifyIBMConnection() method to verify that the Connection Manager connection is still valid (optionally used in step 5 on page F-2), and the releaseIBMConnection() method to return the Connection Manager connection to the pool (used in step 6 on page F-2).

verifyIBMConnection()

```
public boolean verifyIBMConnection()
throws IBMConnMgrException
```

The Connection Manager might take a Connection Manager connection away from a servlet if the Connection Manager connection has been inactive for a specified length of time. The MaximumAge property in the was.conf file is used to specify this time period.

If a servlet uses a DB2 connection for several interactions within one user request, your developers may want to invoke the verifyIBMConnection() method before each interaction to check whether the servlet still owns the associated Connection Manager connection from the pool. If the servlet still owns the connection, invoking the method will also reset a last-used timestamp.

If a servlet's interactions with DB2 (from one user request) will complete within a few seconds, there is probably no need to use the verifyIBMConnection() method — the request will complete long before there is any chance that the Connection Manager will take away the connection.

Returns:

True if the servlet still owns the Connection Manager connection; otherwise false.

releaseIBMConnection()

```
public void releaseIBMConnection()
throws IBMConnMgrException
```

When a servlet no longer needs the Connection Manager connection object, the servlet uses this method to release the connection back to the pool. This should be done at the end of each user request.

Appendix G. Messages EJS3002I - EJS3087E

EJS3002I This is IBM WebSphere Application Server for OS/390 3.50 built on OS/390 Version v Release r, WAS Service Level ll

Explanation: This message identifies WebSphere Application Server's service level and the MVS version and release upon which WebSphere Application Server was built.

User Response: Informational message, no action required.

EJS3003I Built on *ddd* at *ttt*.

Explanation: This message identifies the date and time when WebSphere Application Server was built. *ddd* specifies the date and *ttt* specifies the time.

User Response: Informational message, no action required.

EJS3004I Started at *ttt*

Explanation: This message identifies the time when WebSphere Application Server was started. *ttt* specifies the time.

User Response: Informational message, no action required.

EJS3005I Started Server Type = *nnn*

Explanation: This message identifies WebSphere Application Server's started server type. *nnn* is either STANDALONE, WQ_Daemon, or WQ_Server.

User Response: Informational message, no action required.

EJS3006I Restarted at *ttt*

Explanation: This message specifies the time when WebSphere Application Server restarted. *ttt* is the time.

User Response: Informational message, no action required.

EJS3007E IBM WebSphere Application Server for OS390 native plugin cannot restart...

Explanation: This message indicates WebSphere Application Server cannot restart because an error occurred during termination.

User Response: Examine the trace log for entries pertaining to the termination error. Correct any problem(s) indicated there. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3008I IBM WebSphere Application Server for OS390 native plugin is reinitializing...

Explanation: This message indicates WebSphere Application Server is reinitializing.

User Response: Informational message, no action required.

EJS3009I Started Queue State = *ttt*

Explanation: This message identifies WebSphere Application Server's started queue state. *ttt* is either: STANDALONE, WQ_Daemon, or WQ_Server

User Response: Informational message, no action required.

EJS3010E IBM WebSphere Application Server for OS390 unable to extract the Queue State environment variable

Explanation: This message indicates WebSphere Application Server is unable to extract the Queue State environment variable.

User Response: Examine the error log for entries pertaining to the extract error. Correct any problem(s) indicated there. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3011E IBM WebSphere Application Server for OS390 Unable to extract the APPLENV environment variable

Explanation: This message indicates WebSphere Application Server is unable to extract the APPLENV environment variable.

User Response: Examine the error log for entries pertaining to the extract error. Correct any problem(s) indicated there. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3012E IBM WebSphere Application Server for OS390 Unable to extract the INIT_STRING environment variable

Explanation: This message indicates the HTTP Server's ServerInit directive (to initialize WebSphere Application Server) does not contain the applicationserver_root parameter.

See the section ″Verifying the Application Server installation″ in ″WebSphere Application Server for

OS/390 Standard Edition, Version 3.5: Planning, Installing and Using", GC34–4835, for specific information on the correct way to specify the initialization string.

User Response: Correct the configuration error. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3013E IBM WebSphere Application Server for OS390 unable to parse configuration parameter: ***nnn***

Explanation: This message indicates the HTTP Server's ServerInit directive (to initialize WebSphere Application Server) has an invalid applicationserver_root parameter. That parameter does not begin with a forward slash.

See the section "Verifying the Application Server installation" of "WebSphere Application Server for OS/390 Standard Edition, Version 3.5: Planning, Installing and Using", GC34–4835, for specific information about the correct way to specify the initialization string.

User Response: Correct the configuration error. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3014E IBM WebSphere Application Server for OS390 Unable to parse the INIT_STRING environment variable

Explanation: This message indicates the HTTP Server's ServerInit directive (to initialize WebSphere Application Server) has an initialization parameter which cannot be parsed by WebSphere Application Server.

See the section "Verifying the Application Server installation" of "WebSphere Application Server for OS/390 Standard Edition, Version 3.5: Planning, Installing and Using", GC34–4835, for specific information about the correct way to specify the initialization string.

User Response: Correct the configuration error. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3015E IBM WebSphere Application Server for OS390 JAVA_HOME is not set

Explanation: This message indicates the environment variable JAVA_HOME is not set.

User Response: Correct the error(s) indicated by the message and/or trace log entries. If the problem persists, call IBM product support.

EJS3016E Initialization Error: Failed to load ***xxx***

Explanation: This message indicates WebSphere Application Server is unable to load the was.conf configuration file.

See the section "Verifying the Application Server installation" of "WebSphere Application Server for OS/390 Standard Edition, Version 3.5: Planning, Installing and Using", GC34–4835, for specific information about the correct way to specify the initialization string.

User Response: Correct the configuration error. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3017E IBM WebSphere Application Server for OS390 native plugin initialization failed :-(

Explanation: This message indicates WebSphere Application Server initialization failed.

User Response: Correct any error(s) indicated by previous messages or by trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3018E Initialization Error: Failed to validate properties

Explanation: This message indicates WebSphere Application Server failed to validate initialization properties because of syntax error.

See "WebSphere Application Server for OS/390 Standard Edition, Version 3.5: Planning, Installing and Using", GC34–4835, for the correct property syntax.

User Response: Correct any error(s) indicated by previous messages or by trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3019E Initialization Error: Failed to obtain storage for get default global name

Explanation: This message indicates WebSphere Application Server failed to obtain required storage.

User Response: Increase the amount of storage. Correct any error(s) indicated by previous message(s) and/or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3020I Startup Parameter -- Install Root = ***nnn***

Explanation: This message specifies the WebSphere Application Server's installation root directory, *nnn*.

User Response: Informational message, no action required.

EJS3021I Startup Parameter -- Configuration file = *nnn*

Explanation: This message identifies WebSphere Application Server's configuration file.

User Response: Informational message, no action required.

EJS3022I Startup Parameter -- Default Global properties file = *nnn*

Explanation: This message identifies WebSphere Application Server's default global properties configuration file.

User Response: Informational message, no action required.

EJS3023I Startup Parameter -- JDK install directory (JAVA_HOME) = *nnn*

Explanation: This message identifies WebSphere Application Server's JDK installation directory.

User Response: Informational message, no action required.

EJS3024E Initialization Error: Failed to load Default Global properties file nnn

Explanation: This message indicates WebSphere Application Server failed to load the default global properties file default_global.properties.

See ″Appendix ″C″ ″WebSphere Application Server for OS/390 Standard Edition, Version 3.5: Planning, Installing and Using″, GC34–4835, for the correct way to specify default global properties.

User Response: Correct the configuration error. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3025E Initialization Error: Unable to set product log level

Explanation: This message indicates WebSphere Application Server failed to set the logging level.

See ″WebSphere Application Server for OS/390 Standard Edition, Version 3.5: Planning, Installing and Using″, GC34–4835, for the correct way to specify the product logging level.

User Response: Correct the configuration error. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3026I Startup Parameter -- Plugin Logging Level = *nnn*

Explanation: This message identifies WebSphere Application Server's plugin logging level. *nnn* is either INFO, ERROR, WARNING.

User Response: Informational message, no action required.

EJS3027E Initialization Error: Configuration file version number in *nnn* **is missing or incorrect**

Explanation: This message indicates WebSphere Application Server was.conf configuration file version number property appserver.version is missing or specifies an incorrect value. It must specify: appserver.version=3.50.

See ″WebSphere Application Server for OS/390 Standard Edition, Version 3.5: Planning, Installing and Using″, GC34–4835, for the correct way to specify the appserver.version property in the was.conf configuration file.

User Response: Correct any error(s) indicated by previous messages or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3028E Initialization Error: Unable to obtain storage while setting plugin trace level

Explanation: This message indicates WebSphere Application Server failed to obtain required storage.

User Response: Increase the amount of storage. Correct any error(s) indicated by previous message(s) and/or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3029E Initialization Error: Unable to obtain storage while setting servlet engine trace level

Explanation: This message indicates WebSphere Application Server failed to obtain required storage.

User Response: Increase the amount of storage. Correct any error(s) indicated by previous message(s) and/or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3030E Initialization Error: Unable to obtain storage while reading product work directory

Explanation: This message indicates WebSphere Application Server failed to obtain required storage.

User Response: Increase the amount of storage. Correct any error(s) indicated by previous message(s) and/or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3031E Initialization Error: Unable to obtain storage while reading java eventlib

Explanation: This message indicates WebSphere Application Server failed to obtain required storage.

User Response: Increase the amount of storage. Correct any error(s) indicated by previous message(s) and/or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3032E Initialization Error: setenv("JAVA_PROPAGATE","*nnn*",1) failed, errno=*xx*, __errno2()=*yy*

Explanation: This message indicates WebSphere Application Server's call to setenv failed.

User Response: Correct the error(s) indicated by the errno and errno2 fields, and any error(s) indicated by previous message(s) and/or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3033I Startup Parameter -- Plugin Logging Directory = *nn*

Explanation: This message identifies WebSphere Application Server's plugin logging directory, *nn*.

User Response: Informational message, no action required.

EJS3034I Initialization Error: Unable to obtain storage while creating initialization data

Explanation: This message indicates WebSphere Application Server is unable to obtain required storage.

User Response: Increase the amount of storage. Correct any error(s) indicated by previous message(s) and/or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3035I IBM WebSphere Application Server for OS390 native plugin initialization went OK :-)

Explanation: This message indicates WebSphere Application Server successfully initialized.

User Response: Informational message, no action required.

EJS3036W Warning: Server IP address could not be obtained.

Explanation: This message indicates WebSphere Application Server's IP address could not be obtained.

User Response: Correct any error(s) indicated by previous message(s) and/or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3037W Warning: Failure obtaining storage for server address.

Explanation: This message indicates WebSphere Application Server failed to obtain required storage.

User Response: Increase the amount of storage. Correct any error(s) indicated by previous message(s) and/or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3038I Startup Parameter -- Using server IP address: *xxx*

Explanation: This message identifies WebSphere Application Server's IP address.

User Response: Informational message, no action required.

EJS3039I Error Log Started at *ttt*

Explanation: This message indicates the time (*ttt*) that WebSphere Application Server's error log started recording.

User Response: Informational message, no action required.

EJS3040E Specification violation -- write not called

Explanation: This message is typically symptomatic of an application error. For example, the application did not generate any output.

User Response: Ensure that the failing application catches and reports exceptions. Examine the ncf log for any message(s) associated with the failing application. Correct the application error(s). Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3041E SE specification violation!!! prepare_for_write() was already called

Explanation: This message indicates a WebSphere Application Server internal error.

User Response: Call IBM product support.

EJS3042E SE specification violation!!! prepare_for_write() was not called before write()

Explanation: This message indicates a WebSphere Application Server internal error.

User Response: Call IBM product support.

EJS3043E SE specification violation!!! send_error() was called after prepare_for_write()

Explanation: This message is typically symptomatic of an application error.

User Response: Ensure that the failing application catches and reports exceptions. Examine the ncf log for any message(s) associated with the failing application. Correct the application error(s). Restart WebSphere Application Server. If the problem persists,call IBM product support.

EJS3044E Could not find class nnn ...

Explanation: This message indicates a WebSphere Application Server internal error.

User Response: Call IBM product support.

EJS3045E Initialization Failed: DLL Version Mismatch

Explanation: This message indicates a WebSphere Application Server internal error.

User Response: Call IBM product support.

EJS3046E Initialization Failed: No connection data pointer provided.

Explanation: This message indicates a WebSphere Application Server internal error.

User Response: Cestart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3047E Initialization Failed: No initialization data pointer provided.

Explanation: This message indicates a WebSphere Application Server internal error.

User Response: Call IBM product support.

EJS3048E Initialization Failed: No bootstrap properties provided.

Explanation: This message indicates a WebSphere Application Server internal error.

User Response: Cestart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3049E Initialization Failed: No default server name specified.

Explanation: This message indicates a WebSphere Application Server internal error.

User Response: Call IBM product support.

EJS3050E Initialization Failed: No default server software specified.

Explanation: This message indicates a WebSphere Application Server internal error.

User Response: Call IBM product support.

EJS3051E Initialization Failed: OSE Version mismatch.

Explanation: This message indicates a WebSphere Application Server internal error.

User Response: Call IBM product support.

EJS3052E Initialization Failed: Invalid engine type specified.

Explanation: This message indicates a WebSphere Application Server internal error.

User Response: Call IBM product support.

EJS3053E Failed to Initialize OSE: rc = *nnn*

Explanation: This message indicates WebSphere Application Server failed initialization.

User Response: Correct any error(s) indicated by previous messages or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3054E Failed to Initialize in-process JVM : return code = *nnn*

Explanation: This message indicates WebSphere Application Server failed initialization.

User Response: Correct any error(s) indicated by previous messages or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3055E Failed to create in-process JVM : return code = *nnn*

Explanation: This message indicates WebSphere Application Server failed initialization.

User Response: Correct any error(s) indicated by previous messages or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3056E Verification of in-process JVM failed : return code = *nnn*

Explanation: This message indicates WebSphere Application Server failed initialization.

User Response: Correct any error(s) indicated by previous messages or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3057E Switching Thread Security Failed: Error in userid...

Explanation: This message indicates a WebSphere Application Server internal error.

User Response: Call IBM product support.

EJS3058E Switching Thread Security Failed: Error in password...

Explanation: This message indicates a WebSphere Application Server internal error.

User Response: Call IBM product support.

EJS3059E Switching Thread Security Failed: security create failed - rc= *nn* **errno=** *nn* **errno2 =** *nn*

Explanation: This message indicates a WebSphere Application Server internal error.

User Response: Call IBM product support.

EJS3060E Switching Thread Security Failed: security delete failed - rc= *nn* **errno=** *nn* **errno2 =** *nn*

Explanation: This message indicates a WebSphere Application Server internal error.

User Response: Call IBM product support.

EJS3061E Can't open directory *nnn*

Explanation: This message indicates WebSphere Application Server was unable to open directory *nnn* specified in the application server classpath.

User Response: Ensure that the directory exists and is readable by WebSphere Application Server. Correct any error(s) indicated by previous messages or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3062E Can't find system environment variable [*nnn*]

Explanation: This message indicates the was.conf property appserver.usesystemclasspath was set to true but WebSphere Application Server cannot find the CLASSPATH system environment variable.

User Response: Correct any error(s) indicated by previous messages or trace log entries. Restart WebSphere Application Server.If the problem persists, call IBM product support.

EJS3063E Failed in classpath validation

Explanation: This message indicates WebSphere Application Server failed in classpath validation.

User Response: Correct any error(s) indicated by previous messages or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3064E BM WebSphere Application Server for OS390 detected an illegal value for property *nnn*

Explanation: This message indicates WebSphere Application Server detected an illegal value for property *nnn*.

See ″WebSphere Application Server for OS/390 Standard Edition, Version 3.5: Planning, Installing and Using″, GC34–4835, for a description of property *nnn*.

User Response: Correct any error(s) indicated by previous messages or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3065E IBM WebSphere Application Server for OS390 detected illegal property *nnn*

Explanation: This message indicates WebSphere Application Server detected an illegal value for property *nnn*.

See ″WebSphere Application Server for OS/390 Standard Edition, Version 3.5: Planning, Installing and Using″, GC34–4835, for a description of property *nnn*.

User Response: Correct any error(s) indicated by previous messages or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3066E Initialization Error: Failed to obtain storage for ini file

Explanation: This message indicates WebSphere Application Server failed to obtain required storage.

User Response: Increase the amount of storage. Correct any error(s) indicated by previous messages or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3067E Initialization Error: Failed to open file ***nnn***

Explanation: This message indicates a WebSphere Application Server internal error.

User Response: Call IBM product support.

EJS3068E Error : wrong input to ***nnn***

Explanation: This message indicates a WebSphere Application Server internal error.

User Response: Call IBM product support.

EJS3069E Error : in ***nnn*****, property** ***nnn*** **is missing**

Explanation: This message indicates a WebSphere Application Server internal error.

User Response: Call IBM product support.

EJS3070E Error : in ***nnn*****, given buffer is short**

Explanation: This message indicates a WebSphere Application Server internal error.

User Response: Call IBM product support.

EJS3071E Error : obtaining storage for plugin libpath

Explanation: This message indicates WebSphere Application Server failed to obtain required storage.

User Response: Increase the amount of storage. Correct any error(s) indicated by previous messages or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3072E errno: ***nn*****, __errno2():** ***nnn*****, loading DLL [*****nnn*****]:** ***nnn***

Explanation: This message indicates WebSphere Application Server failed loading DLL *nnn*.

User Response: Correct any error(s) indicated by the errno and errno2 fields, and correct any error(s) indicated by previous messages or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3073E Loading function pointer for [***nnn*****] from [*****nnn*****]: errno:** ***nnn*** **__errno2():** ***nnn***

Explanation: This message indicates WebSphere Application Server failed loading function pointer for *nnn*.

User Response: Correct any error(s) indicated by the errno and errno2 fields, and any error(s) indicated by previous messages or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3074E errno: ***nnn*****, __errno2():** ***nnn*****, in** ***nnn*** **can not get the ose lib [*****nnn*****]**

Explanation: This message indicates WebSphere Application Server failed getting ose lib *nnn*.

User Response: Correct the error(s) indicated by the errno and errno2 fields, and any error(s) indicated by previous messages or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3075E Error : in ***nnn*** **can not get the ose router lib**

Explanation: This message indicates a WebSphere Application Server internal error.

User Response: Call IBM product support.

EJS3076E Error - NULL property file name

Explanation: This message indicates a WebSphere Application Server internal error.

User Response: Call IBM product support.

EJS3077E Error - Couldn't find property file ***nnn***

Explanation: This message indicates a WebSphere Application Server internal error.

User Response: Call IBM product support.

EJS3078I Generated CLASSPATH follows:

Explanation: This message indicates WebSphere Application Server's generated classpath follows.

User Response: Informational message, no action required.

EJS3079I CLASSPATH entry: ***nnn***

Explanation: This message indicates WebSphere Application Server's classpath entry.

User Response: Informational message, no action required.

EJS3080I End of generated CLASSPATH

Explanation: This message indicates WebSphere Application Server's generated classpath echo has ended.

User Response: Informational message, no action required.

EJS3081E Error : in *nnn*, *nnn* is not a directory

Explanation: This message indicates that WebSphere Application Server, while validating properties, detected *nnn* is not a directory.

See ″WebSphere Application Server for OS/390 Standard Edition, Version 3.5: Planning, Installing and Using″, GC34–4835, for a description of the correct way to specify was.conf file properties.

User Response: Correct any error(s) indicated by previous messages or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3082E Error : in *nnn* no java libpath given in *nnn*.

Explanation: This message indicates a WebSphere Application Server internal error.

User Response: Call IBM product support.

EJS3083E Error : in *nnn*, can not generate directory from *nnn* and *nnn*

Explanation: This message indicates a WebSphere Application Server internal error.

User Response: Call IBM product support.

EJS3084E Detected bad classpath entry: *nnn*

Explanation: This message indicates WebSphere Application Server detected bad classpath entry *nnn*.

User Response: Correct any error(s) indicated by previous messages or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

EJS3085E Missing property [*nnn*] using default [*nnn*]

Explanation: This message indicates WebSphere Application Server detected missing property *nnn* and is using the default of *nnn*.

User Response: Informational message, no action required.

EJS3086W Warning: Default Global property [*nnn*]: *nnn*

Explanation: This message indicates WebSphere Application Server is using the value *nnn* for Default Global property *nnn*.

User Response: Informational message, no action required.

EJS3087E Error: Invalid Default Global property [*vvv*]: *nnn*

Explanation: This message indicates WebSphere Application Server detected an invalid value (vvv) for Default Global property *nnn*.

See ″WebSphere Application Server for OS/390 Standard Edition, Version 3.5: Planning, Installing and Using″, GC34–4835, for a description of the correct way to specify default.global file properties.

User Response: Correct any error(s) indicated by previous messages or trace log entries. Restart WebSphere Application Server. If the problem persists, call IBM product support.

Appendix H. Apache Software License, Version1.1

Glossary

All technical terms and abbreviations used in WebSphere Application Server documentation are contained in the publication *IBM Glossary of Computing Terms* located at URL:

http://www.ibm.com/networking/nsg/nsgmain.htm.

Bibliography

This bibliography lists the books related to Version 3.5 of the WebSphere Application Server for OS/390.

Application Server books are available from the WebSphere Application Server for OS/390 Web site at URL:

```
http://www.ibm.com/software/websphere/
  appserv/zos_os390/library.html
```

For a summary of available OS/390 books and online information, see the *OS/390 Information Road Map* which is available in BookManager format on the OS/390 CD-ROM Collection Kit and from the OS/390 Web site at URL:

```
http://www.ibm.com/s390/os390/bkserv/
```

Application Server publications

- *Application Server Planning, Installing, and Using*, GC34–4835

  This book contains the information you need to plan for, install, configure, and use the Application Server.
- *WebSphere Troubleshooter for OS/390*. This document is only available on the Web and provides the most current hints and tips (debugging, tuning, and browser) for the Application Server and the Web server. To access the *Troubleshooter*, go to URL:

  ```
  http://www.ibm.com/software/websphere/
     httpservers/troubleshooter.html
  ```

Web server publications

Web server documentation can be accessed from the default Front Page of your Web server.

For the most current documentation and updates, go to the Web site Library page for your Web server. For more information, see "Required OS/390 Web server" on page 1-1.

OS/390 publications

- *OS/390 Information Road Map*, GC28-1727

  This book describes available information for the elements and features in OS/390. It also explains how to order OS/390 documentation and how to access online OS/390 information.
- *OS/390 Planning for Installation*, GC28-1726

  This book lists the elements and features in OS/390. It explains how to get OS/390 up and running, and provides information about migration actions for specific elements of OS/390.
- *OS/390 MVS Planning: Workload Management*, GC28-1761

  This book explains Workload Management (WLM) concepts and interfaces, and includes the steps required for using WLM as well as its benefits.

Notices

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

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

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

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

This product includes software developed by the Apache Software Foundation (http://www.apache.org/). The relevant terms and conditions, notices and other information regarding this software is provided in Appendix H, "Apache Software License, Version1.1", on page H-1.

This publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of the Application Server.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

Index

Program Number: 5655–A98

Printed in U.S.A.

GC34-4835-05