WebSphere® Application Server V4.0.1 for z/OS and OS/390

IBM

Assembling Java™ 2 Platform, Enterprise Edition (J2EE™) Applications

SA22-7836-06

WebSphere® Application Server V4.0.1 for z/OS and OS/390

IBM

Assembling Java™ 2 Platform, Enterprise Edition (J2EE™) Applications

SA22-7836-06

Note

Before using this information and the product it supports, be sure to read the general information under Appendix E, “Notices,” on page 401.

Seventh Edition (June 2003)

This is a major revision of SA22–7836–05.

This edition applies to WebSphere Application Server V4.0.1 for z/OS and OS/390 (5655-F31), and to all subsequent releases and modifications until otherwise indicated in new editions.

The most current versions of the WebSphere Application Server V4.0.1 for z/OS and OS/390 publications are at this Web site: `http://www.ibm.com/software/webservers/appserv/zos_os390/`

© Copyright International Business Machines Corporation 2000, 2003. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 2000, 2003

Figures

Tables

About this book

WebSphere Application Server V4.0.1 for z/OS and OS/390: Assembling J2EE Applications, SA22-7836 describes how to create, assemble, and install Java 2 Enterprise Edition (J2EE) applications to run in the WebSphere V4.0.1 for z/OS and OS/390 environment. These J2EE applications may consist of Enterprise Java beans, Java servlets and JavaServer Pages (JSPs). WebSphere for z/OS J2EE servers provide an application environment that allows these applications to be highly managed and integrated with databases and transactional systems on z/OS or OS/390.

Note: The full product name is WebSphere Application Server V4.0.1 for z/OS and OS/390, referred to in this text as WebSphere for z/OS.

For information about migrating J2EE applications from one of the following, see *WebSphere Application Server V4.0.1 for z/OS and OS/390: Migration*, GA22-7860:
- Previous releases of WebSphere for z/OS,
- Previous versions of WebSphere Application Server for z/OS and OS/390, or
- WebSphere Application Server products for distributed platforms (such as Windows NT).

Who should read this book

This book is intended primarily for programmers who fulfill the tasks defined in the Sun Microsystems Java 2 Enterprise Edition Specification V1.2 for the roles of Application Component Provider, Application Assembler, and Deployer. For details about those roles and associated responsibilities, as well as additional overview information about J2EE, refer to the Sun Microsystems J2EE specification, which is available at:

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

Where to find related information, tools, and supplements

This is a list of books that are in the WebSphere for z/OS library. They can be found by accessing the following Web site:

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

- *WebSphere Application Server V4.0.1 for z/OS and OS/390: Program Directory*, GI10-0680, describes the elements of and the installation instructions for WebSphere for z/OS.
- *WebSphere Application Server V4.0.1 for z/OS and OS/390: License Information*, LA22-7855, describes the license information for WebSphere for z/OS.
- *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834, describes the planning, installation, and customization tasks and guidelines for WebSphere for z/OS.
- *WebSphere Application Server V4.0.1 for z/OS and OS/390: Messages and Diagnosis*, GA22-7837, provides diagnosis information and describes messages and codes associated with WebSphere for z/OS.
- *WebSphere Application Server V4.0.1 for z/OS and OS/390: Operations and Administration*, SA22-7835, describes system operations and administration tasks.

- *WebSphere Application Server V4.0.1 for z/OS and OS/390: Assembling J2EE Applications*, SA22-7836, describes how to develop, assemble, and install J2EE applications in a WebSphere for z/OS J2EE server.
- *WebSphere Application Server V4.0.1 for z/OS and OS/390: Assembling CORBA Applications*, SA22-7848, describes how to develop, assemble, and deploy CORBA applications in a WebSphere for z/OS (MOFW) server.
- *WebSphere Application Server V4.0.1 for z/OS and OS/390: System Management User Interface*, SA22-7838, describes the system administration and operations tasks as provided in the Systems Management User Interface.
- *WebSphere Application Server V4.0.1 for z/OS and OS/390: System Management Scripting API*, SA22-7839, describes the functionality of the WebSphere for z/OS Systems Management Scripting API product.
- *WebSphere Application Server V4.0.1 for z/OS and OS/390: Migration*, GA22-7860, describes migration procedures for WebSphere for z/OS.

Here are some other WebSphere Application Server books on that Web site that you might find particularly helpful:

- *WebSphere Application Server for OS/390 V3.5 Standard Edition Planning, Installing, and Using*, GC34-4835, provides information about running the Version 3.5 runtime shipped with the V4.0.1 product within the HTTP Server address space. You can use this configuration if you want to continue running non-J2EE-compliant Web applications in the V3.5 runtime within the HTTP Server address space while migrating to the full WebSphere for z/OS run time.
- *Building Business Solutions with WebSphere*, SC09-4432

The integrated WebSphere Application Server Advanced Edition and WebSphere Application Server Enterprise Edition InfoCenter includes CORBA (MOFW) information you need to code CORBA (MOFW) components. Go to:

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

For additional WebSphere for z/OS tools and supplements, go to the following Web site and select the download link:

```
http://www.ibm.com/software/webservers/appserv/zos_os390/
```

You might also need to refer to information about other z/OS or OS/390 elements and products. All of this information is available through links at the following Internet locations:

```
http://www.ibm.com/servers/eserver/zseries/zos/
http://www.ibm.com/servers/s390/os390/
```

How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information. You can e-mail your comments to:

```
wasdoc@us.ibm.com
```

or fax them to 919-254-0206.

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

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

Summary of changes

Summary of changes
for SA22-7836-06
WebSphere for z/OS V4.0.1
as updated, July 2003
service level W401508

This book contains information previously presented in SA22-7836-05, which supports WebSphere for z/OS. The following is a summary of changes to this information:

- Information about setting access intent for custom finders (APAR PQ69991) has been updated in:
  - "Checklist for using pessimistic concurrency control" on page 58, and
  - "Developing Enterprise beans" on page 117
- Information about setting RunAs identity in "Determining the user ID for resource authentication" on page 73 has been updated (APAR PQ71760).
- Information about setting file permissions in "Steps for creating JCL procedures for the control and server regions" on page 146 has been updated (APAR PQ67872 PTF UQ74000, service level W401500).
- Information about required software in "Application clients that run on non-z/OS platforms" on page 217 has been updated (APAR PQ67662, PTF UQ72838, service level W401407).
- Appendix A, "Environment and JVM properties files," on page 299 contains new or changed descriptions of the following environment variables and JVM properties:
  - `com.ibm.ejs.EJBCache.size` (APAR PQ70308)
  - `com.ibm.websphere.cmp.cache.maxLevels` (APAR PQ69293)
  - `com.ibm.websphere.cmp.cache.printlevels` (APAR PQ69293)
  - `com.ibm.websphere.persistence.bean.managed.custom.finder.access.intent` (APAR PQ69991)
  - `com.ibm.websphere.preconfiguredCustomServices`
  - `IBM_JVM_ST_VERBOSEGC_LOG` (APAR PQ69792)
  - `service.debug.enabled` (APAR PQ70287)
  - `invocationCacheSize` (APAR PQ71648)
  - `RECYCLE_J2EE_SERVERS` (APAR PQ68957)
- WebSphere HTTP Plug-in for z/OS information has been added to Chapter 4, "A closer look at the J2EE server," on page 19 and Chapter 8, "Creating a J2EE server run-time environment," on page 143 (APAR PQ68250).
- Custom user registry information has been added to Chapter 4, "A closer look at the J2EE server," on page 19 and Chapter 8, "Creating a J2EE server run-time environment," on page 143 (PTF UQ71162).
- A description of the new jdbcconnpool.<pool-name>.provider was.conf file property, that can be used to specify the JDBC database management system (DBMS) that is hosting the Application Server connection pools, was added to Appendix C, "Using the Alternate Configuration Option," on page 365.
- A description of the code change that might need to be made to a Web service ported from a WebSphere Application Server for distributed platforms product

© Copyright IBM Corp. 2000, 2003

in order to use the externalConfigURL tag to pass initialization parameters has been added to Chapter 15, “Creating and deploying Web Services,” on page 249 (APAR PQ73188).

- Additional information has been included in the first two notes contained in Figure 5. These notes describe how to enable Form-Based authentication or single sign-on capability.
- The description of how to run the JspBatchCompiler.sh script has been updated.
- The session.enable property has been removed from Appendix B, “Default webcontainer.conf file,” on page 351 because it is no longer supported. Session must always be enabled.
- Two new sections, “Properties of WebSphere plug-ins for Web servers” on page 183 and “Installing a Web server plug-in on a Microsoft Internet Information Server (IIS)” on page 183, were added to Chapter 8.
- A description of how to host an application containing a Java AWT class has been added to Chapter 2, “Overview of application development and tools,” on page 9 (APAR PQ69795).
- A description of the code changes you may have to make to a Web service you are porting from the distributed platform version of the product to the z/OS version has been added to “Deploying an Enterprise application as a SOAP-accessible Web Service” on page 250.

Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Summary of changes
for SA22-7836-05
WebSphere for z/OS V4.0.1
as updated, September 2002
service level W401400

This book contains information previously presented in SA22-7836-04, which supports WebSphere for z/OS. The following is a summary of changes to this information:

- Information has been added to Chapters 4 and 8 to describe the new functions provided in PTFs UQ70037, UQ90051 and UQ90052. Updates include descriptions of how to:
  - Maintain session data in-memory across multiple server instances.
  - Use Version 2 of DB2 Session Persistence.
  - Use dynamic fragment caching.
  - Use the WebSphere plug-ins for Web servers.
  - Set up client certificates for use with the HTTP Transport Handler.
  - Pre-compile JSPs.

  The information about maintaining session data in memory across server instances was previously presented in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Maintaining Session Affinity Across Multiple Server Instances*.
- Additional information on Form-based Login was added to “Authenticating Web Clients” on page 37.
- A new topic, “Overview of SQLID for managed datasources” on page 34, has been added to Chapter 4, “A closer look at the J2EE server,” on page 19 to describe WebSphere for z/OS support for DB2 secondary authorization IDs to control the use of unqualified table references in container-managed (CMP) and bean-managed (BMP) Enterprise beans (APARs PQ65206, PQ65207, and PQ66463; PTFs UQ70037, UQ90051 and UQ90052; Service level W401400).

- A new section, “Running applications developed in WebSphere Studio Application Developer Integration Edition” on page 76, has been added for the WebSphere Studio Application Developer Integration Edition for z/OS Connectors (APARs PQ65206, PQ65207, and PQ66463; PTFs UQ70037, UQ90051 and UQ90052; Service level W401400). In addition, throughout the book, many of the references to VisualAge for Java have been either replaced with, or modified with the addition of a more current development tool: WebSphere Studio Application Developer Integration Edition.
- The following restriction has been removed from the section ″Creating Enterprise beans″ in Chapter 6 because this support is now available if PTFs UQ70037, UQ90051 and UQ90052 have been applied to your system:

  The function which allows JNDI lookups from auto-started servlets does not support the use of transactions from within servlet init() methods. The use of any EJB which requires a transaction environment, such as TX_REQUIRED or TX_MANDATORY, is not supported when the EJB is driven from within a servlet init() method.
- A new topic, “Preparing applications for assembly and installation” on page 120, describes WebSphere for z/OS class loaders and their operation, and how that operation might affect application packaging (APARs PQ65206, PQ65207, and PQ66463; PTFs UQ70037, UQ90051 and UQ90052; Service level W401400). It also contains guidelines for selecting a classloader mode, packaging applications, and collecting trace data about classloader operation. This information replaces the guidelines previously published in topics entitled *Packaging beans in JAR files*, and *Packaging Web components in WAR files*.
- A new section, “Direct Deployment Tool/390fy” on page 141, has been added for the Direct Deployment Tool/390fy (APARs PQ65206, PQ65207, and PQ66463; PTFs UQ70037, UQ90051 and UQ90052; Service level W401400).
- Two topics related to defining a J2EE server have been updated with instructions for setting permission bits for files created by applications (APAR PQ60198):
  – “Steps for completing manual z/OS or OS/390 tasks” on page 144, and
  – “Steps for creating JCL procedures for the control and server regions” on page 146..
- Appendix A, “Environment and JVM properties files,” on page 299 contains updated information about managing JVM properties, which includes information previously presented in Washington Systems Center FAQ FQ101947.
- Appendix A, “Environment and JVM properties files,” on page 299 contains descriptions of the following new environment variables and JVM properties:
  – `APP_EXT_DIR`
  – `BBOC_HTTP_MODE`
  – `BBOC_HTTP_SSL_CBIND`
  – `BBOC_HTTP_SSL_MODE`
  – `BBOC_HTTPALL_NETWORK_QOS`
  – `BBOC_HTTPALL_TCLASS_FILE`
  – `BBOO_ACCEPT_HTTP_WORK_AFTER_N_SECS` (APAR PQ63711)
  – `BBOO_ACCEPT_HTTP_WORK_AFTER_N_SRS` (APAR PQ63711)
  – `CLONEID`
  – `com.ibm.websphere.cmp.connection.policy`
  – `com.ibm.websphere.cmp.`*`Location Name`*`.connection.timeout`
  – `com.ibm.websphere.cmp.`*`Location Name`*`.preparedStatement.poolsize`
  – `com.ibm.websphere.cmp.`*`Location Name`*`.connection.maxTrans`
  – `com.ibm.websphere.preconfiguredCustomServices`

– `com.ibm.websphere.sendredirect.compliance`
– `com.ibm.ws.classloader.ejbDelegationMode`
– `com.ibm.ws.classloader.J2EEApplicationMode`
– `com.ibm.ws.classloader.warDelegationMode`
– `com.ibm.ws390.wc.config.dynxmlfilename`
– `com.ibm.ws390.wc.config.dynsrvxmlfilename`
– `IIOP_SERVER_SESSION_KEEPALIVE` (APAR PQ62464)
– `TRACESPECIFIC`

- Many of the references to the WebSphere for z/OS Application Assembly tool have been changed to "a WebSphere application assembly tool", now that you are no longer required to use the z/OS edition of the tool except when you are developing applications that use certain IBM extensions (APARs PQ65206, PQ65207, and PQ66463; PTFs UQ70037, UQ90051 and UQ90052; Service level W401400). "A WebSphere application assembly tool" means either the z/OS edition of the Application Assembly tool, the WebSphere Studio Application Developer tool, the WebSphere Studio Application Developer Integration Edition tool, or Direct Deployment Tool/390fy.

Summary of changes
for SA22-7836-04
WebSphere for z/OS V4.0.1
as updated, July 2002
service level W401082

This book contains information previously presented in SA22-7836-03, which supports WebSphere for z/OS. The following is a summary of changes to this information:

- Information in "Connectors" on page 64 has been clarified.
- Information has been added to describe the new support for trust associations between third party servers and WebSphere for z/OS (APAR PQ55181, PTF UQ90049, Service Level 11). Updates include:
  – A description of the trust association interceptor support in "Web Security" on page 36 and "The WebSphere for z/OS environment for Web applications" on page 80.
  – The procedure to enable the trust association interceptor support in "Steps for configuring trust association" on page 202.
  – The new JVM property that enables Web security in Appendix A, "Environment and JVM properties files," on page 299.
  – The new webcontainer.conf file properties that define a trust association interceptor to WebSphere for z/OS in Appendix B, "Default webcontainer.conf file," on page 351.

  This information was previously presented in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Trust Association Interceptor Support*.
- Information about the WebSphere for z/OS server configuration has been updated to include the new HTTPS Transport Handler (APAR PQ59911, PTF UQ90049, Service Level 11). Updates appear in:
  – Chapter 1, "Overview of the WebSphere for z/OS J2EE server," on page 3.
  – "The WebSphere for z/OS environment for Web applications" on page 80.
  – "Steps for adding the J2SERV server" on page 151.
  – Appendix B, "Default webcontainer.conf file," on page 351.

This information was previously presented in *WebSphere Application Server V4.0.1 for z/OS and OS/390: HTTPS Transport Handler*.

- A new chapter has been added to provide information about the new support for Type 4 JDBC Connectors (APAR PQ61755, PTF UQ90050, Service Level W401076). This information includes:
  - Procedures for defining a Type 4 JDBC Connector to WebSphere for z/OS.
  - A procedure for developing and deploying WebSphere for z/OS applications that use Type 4 JDBC connectors.
  - A sample application and other supporting files for using Type 4 JDBC connectors.

  This new information, which appears in Chapter 16, "Using Type 4 JDBC Connectors with WebSphere for z/OS," on page 257, was previously presented in *Using Type 4 JDBC Connectors*.
- Appendix A, "Environment and JVM properties files," on page 299 contains new environment variables and JVM properties, including:

  com.ibm.websphere.preconfiguredCustomServices
  Specifies customer-provided services to be installed in the Java virtual machine that runs under the J2EE server (APAR PQ55873, PTF UQ99329, service level W401030). For information about custom services, see the topic about developing custom services in the InfoCenter for WebSphere Application Server Advanced Edition Version 4.0. The InfoCenter is available at `http://www.ibm.com/software/webservers/appserv/`

Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Summary of changes for SA22-7836-03 WebSphere for z/OS V4.0.1 as updated, March 2002 service level W401038

This book contains information previously presented in SA22-7836-02, which supports WebSphere for z/OS. The following is a summary of changes to this information:

- Information about the new HTTP Session Affinity support (APAR PQ57888, PTF UQ64031, service level L00PTF06), and other changes related to the Web application environment, has been added. These additions and changes include:
  - The HTTP Transport Handler is now presented as your primary HTTP protocol catcher in the concepts in Chapter 1, "Overview of the WebSphere for z/OS J2EE server," on page 3 and "The WebSphere for z/OS environment for Web applications" on page 80.
  - Concepts and instructions for storing session data in-memory now appear in "HTTP session support" on page 87 and "Steps for configuring HTTP Session Support" on page 160, respectively.
  - Changes to properties and other reference material appear in Appendix B, "Default webcontainer.conf file," on page 351:
    - The **session.cookie.path** property is no longer configurable, and has been deleted from the description of the default webcontainer.conf file. The cookie path is automatically set to the context root of the Web application.
    - The **session.datasourcename** property is no longer configurable, and has been deleted from the description of the default webcontainer.conf file. The HTTP session datasource is now defined as a J2EE Resource using the WebSphere for z/OS Administration application.

- The default value for the **session.cookie.name** property has changed to **JSESSIONID**.
- Appendix C, “Using the Alternate Configuration Option,” on page 365 consolidates information on how to set up an IBM HTTP Server and the V3.5 runtime provided with the V4.0.1 product (instructions formerly in Chapter 8, “Creating a J2EE server run-time environment,” on page 143), to host Web applications that were previously run on the V3.5 Standard Edition product. This appendix previously documented only the default V4.0.1 `was.conf` file.

- Information about WebSphere for z/OS-supported connectors (APAR PQ55873, PTF UQ99329, service level W401030), which was previously presented in *WebSphere Application Server V4.0.1 for z/OS and OS/390: WebSphere for z/OS-Supported Connectors*, has been added to this book. This information, in “Connectors” on page 64, includes application assembly and deployment instructions, and guidelines for the use of connectors and connection management policies.
- Information previously presented in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Concurrency Control Management* (APAR PQ55873, PTF UQ99329, service level W401030), has been added to this book. This information, which appears in “IBM Extensions” on page 53, includes:
  - A new section to help application assemblers decide which approach to use for concurrency control, pessimistic or optimistic, for Enterprise beans that use container-managed persistence (CMP beans).
  - A new section to describe the IBM deployment descriptor extension, isolation level, which is a setting related to concurrency control.
- A new section, “Considerations for test and production environments” on page 105,, provides a brief introduction to test and production systems (APAR PQ55866, PTF UQ99328, service level W401014).
- Information about JNDI caching behavior has been clarified in:
  - “Java Naming and Directory Interface™ (JNDI)” on page 43, and
  - Chapter 10, “Using JNDI look-ups,” on page 227.

  WebSphere for z/OS uses JNDI caching by default, but its behavior may be modified by Java client programs only. Your installation cannot modify JNDI caching behavior on a J2EE server level.
- Chapter 15, “Creating and deploying Web Services,” on page 249 contains updated software requirements, and minor updates to examples and terminology.
- Appendix A, “Environment and JVM properties files,” on page 299 contains new environment variables and JVM properties, including:

  com.ibm.CORBA.iiop.noLocalCopies
  Determines whether objects passed between Enterprise beans running in the same JVM are passed by reference instead of by value. (APAR PQ57189, Service Level W401019)

  com.ibm.ws390.wc.serverCheckInterval
  Indicates how frequently, in minutes, the WebSphere for z/OS V3.5 runtime should check the list of servers specified on the com.ibm.ws390.wc.includeWebContainers JVM property to determine if new servers have been added with which is is now authorized to communicate. (PTF UQ61610, service level L00PTF03)

  com.ibm.ws390.wc.includeWebContainers
  Specifies the installed J2EE Servers with which the WebSphere for z/OS V3.5 runtime can communicate. (PTF UQ61610, service level L00PTF03)

com.ibm.ws390.server.classloadermode
Specifies the visibility mode to use for the J2EE server. (PTF UQ61610, service level L00PTF03)

WS_EXT_DIRS
Specifies the common JAR files and directories that can be accessed by multiple applications running in the same J2EE server instance. (PTF UQ63580, service level L00PTF05)

- References to VisualAge for Java and WebSphere Studio as recommended application development tools have been changed to the new IBM WebSphere Studio Application Developer product, which can be used to develop both Enterprise beans and Web applications.

Summary of changes for SA22-7836-02 WebSphere for z/OS V4.0.1 as updated, October 2001

This book contains information previously presented in SA22-7836-01, which supports WebSphere for z/OS. The following is a summary of changes to this information:

- The information about migrating applications has been removed from this book. Information about migrating applications now appears in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Migration*, GA22-7860.
- Chapter 1, "Overview of the WebSphere for z/OS J2EE server," on page 3 lists updated versions or specification levels of Java 2™, Enterprise Edition (J2EE™) technologies and application programming interfaces (APIs) that are supported by WebSphere for z/OS.
- A new section, Chapter 4, "A closer look at the J2EE server," on page 19, has been added to highlight some of the J2EE technologies and APIs that WebSphere for z/OS supports, and to explain key concepts, terminology, and procedures.
- "Security" on page 19 presents an introduction to security mechanisms available in the WebSphere for z/OS environment (including updates introduced through APAR PQ53621, service level W401004).
- Procedures for setting up the WebSphere for z/OS environment for Web applications have been updated. These step-by-step instructions appear in "Steps for enabling J2EE server support for Web applications (optional)" on page 147.
- A new section, Part 3, "Programming and deployment scenarios for J2EE applications," on page 225, has been added to illustrate how to exploit some of the J2EE technologies and APIs that WebSphere for z/OS supports. Some of these topics or scenarios span the coding, assembly, and installation tasks that application assemblers, deployers, and system administrators perform in the WebSphere for z/OS environment.

  These new topics or scenarios include:
  - Chapter 11, "Using security roles and RunAs identities with Enterprise beans," on page 231
  - Chapter 15, "Creating and deploying Web Services," on page 249
- "Debugging and tracing distributed applications" on page 281 describes how to use the IBM Distributed Debugger and Object Level Trace, which provides debugging and tracing capabilities for J2EE application components and their Java clients, which may reside on platforms other than z/OS or OS/390.

Summary of changes for SA22-7836-01

WebSphere for z/OS as updated, June 2001 service level W400018

This book contains information previously presented in SA22–7836-00, which supports WebSphere for z/OS. The following is a summary of changes to this information:

- In various topics in Part 2, "Creating, assembling and deploying J2EE server applications," on page 107, the maximum length of security role names has been corrected. The maximum length is 246 characters.
- The topic "Steps for installing a J2EE application" on page 154 has been updated to include instructions for replacing the resource reference `ws390rt/cmp/jdbc/CMPDS` with a valid datasource for backing entity beans that use container-managed persistence (CMP).
- The following sections contain information that was previously available through the document *WebSphere V4.0 for z/OS or OS/390: Enabling Web Applications on a J2EE server*:
  - "Steps for enabling J2EE server support for Web applications (optional)" on page 147
  - Appendix B, "Default webcontainer.conf file," on page 351
- The information in Chapter 9, "Creating and running WebSphere for z/OS client applications," on page 217 has been clarified, including updates introduced through APAR PQ49461 (PTF UQ54982, service level W400017):
  - The initial JNDI context factory property setting is now `com.ibm.websphere.naming.WsnInitialContextFactory`
  - J2EE application clients must specify the `javax.naming.provider.url` property to access the WebSphere for z/OS naming service on another sysplex, or to access the JNDI on an Advanced Edition WebSphere running on a workstation platform.
- The information in "Logging messages and trace data for Java applications" on page 284 has been changed to reflect the following behavior, introduced through APAR PQ47682 (PTF UQ53715, service level W400010):

  All messages that your application issues will appear in the CTRACE data set for WebSphere for z/OS. Some messages also will appear on the master console or in the error log, depending on the message type:
  - `TYPE_INFORMATION` (or `TYPE_INFO`) will appear on the master console.
  - `TYPE_ERROR` (or `TYPE_ERR`) will appear in the error log.

  Note that comments in the sample code in section "Steps for coding your Java application to issue messages and trace requests" on page 289 also have changed to reflect the changed destinations for messages.

Part 1. Introducing the WebSphere for z/OS J2EE server

© Copyright IBM Corp. 2000, 2003

Chapter 1. Overview of the WebSphere for z/OS J2EE server

WebSphere Application Server for z/OS and OS/390 provides a highly available, secure, reliable, and scalable run-time environment for Java 2 Enterprise Edition (J2EE) applications. This WebSphere for z/OS run-time includes servers for both J2EE and CORBA applications, through its J2EE server and managed-object framework (MOFW) servers, respectively.

The primary focus for WebSphere for z/OS, however, is its J2EE server, which supports both Enterprise JavaBeans and Web components that conform to the J2EE specifications and packaging standards published by Sun Microsystems. These two types of J2EE application components run in a WebSphere for z/OS J2EE server, and can use both:

- The application programming interfaces (APIs) and services that the Java 2 Standard Edition (J2SE) Software Development Kit (SDK) V1.3 provides, and
- Enterprise services such as Java Database Connectivity (JDBC), Java Naming and Directory Interface (JNDI), and the Java Transaction Service (JTS) and API (JTA).

The J2EE specifications dictate which APIs and services each type of application component may use, and the environment in which they must run. Although both Enterprise beans and Web applications may run in a single WebSphere for z/OS J2EE server, each component actually runs in a separate type of container within the J2EE server. Enterprise beans run in the EJB container, and Web applications run in a Web container. These two containers in the WebSphere for z/OS J2EE server conform to the J2EE specifications for run-time environments.

Current WebSphere for z/OS support for J2EE application components includes the J2EE technologies and APIs listed in Table 1.

Table 1. Current WebSphere for z/OS support for J2EE technologies

J2EE technology	Support in WebSphere for z/OS J2EE server	Supported for this type of J2EE application component:	
		Web component	Enterprise bean
Java 2 Standard Edition (J2SE) Software Development Kit (SDK)	V1.3	Yes	Yes
Enterprise JavaBeans	V1.1 (but also supports V1.0 specification)	EJB client API only	Yes
Java servlets	V2.2 specification	Yes	No
JavaServer Pages (JSPs)	V1.1 specification	Yes	No
Java Transaction Service (JTS) and API (JTA)	JTS V1.0 and JTA V1.0.1 supported with distributed transactions	Yes	Yes
Java Database Connectivity (JDBC)	JDBC V2.1 and JDBC Standard Extensions V2.0 (JDBC V1.x is supported for compatibility)	Yes	Yes
Java Naming and Directory Interface (JNDI)	V1.2.2	Yes	Yes

© Copyright IBM Corp. 2000, 2003

Table 1. Current WebSphere for z/OS support for J2EE technologies (continued)

J2EE technology	Support in WebSphere for z/OS J2EE server	Supported for this type of J2EE application component:	
		Web component	Enterprise bean
Java Remote Method Invocation (RMI)	V1.0	Yes	Yes
RMI/IIOP	V1.2.2	Yes	Yes
Java IDL	Supported		
Java Message Service (JMS)	V1.1	Yes	Yes
JavaBeans Activation Framework (JAF)	V1.0.1	Yes	Yes
JavaMail	V1.0.1	Yes	Yes

The WebSphere for z/OS J2EE server supports a variety of client applications that use Enterprise beans to access system resources on z/OS or OS/390. These client applications may run in different environments, as shown in Figure 1.

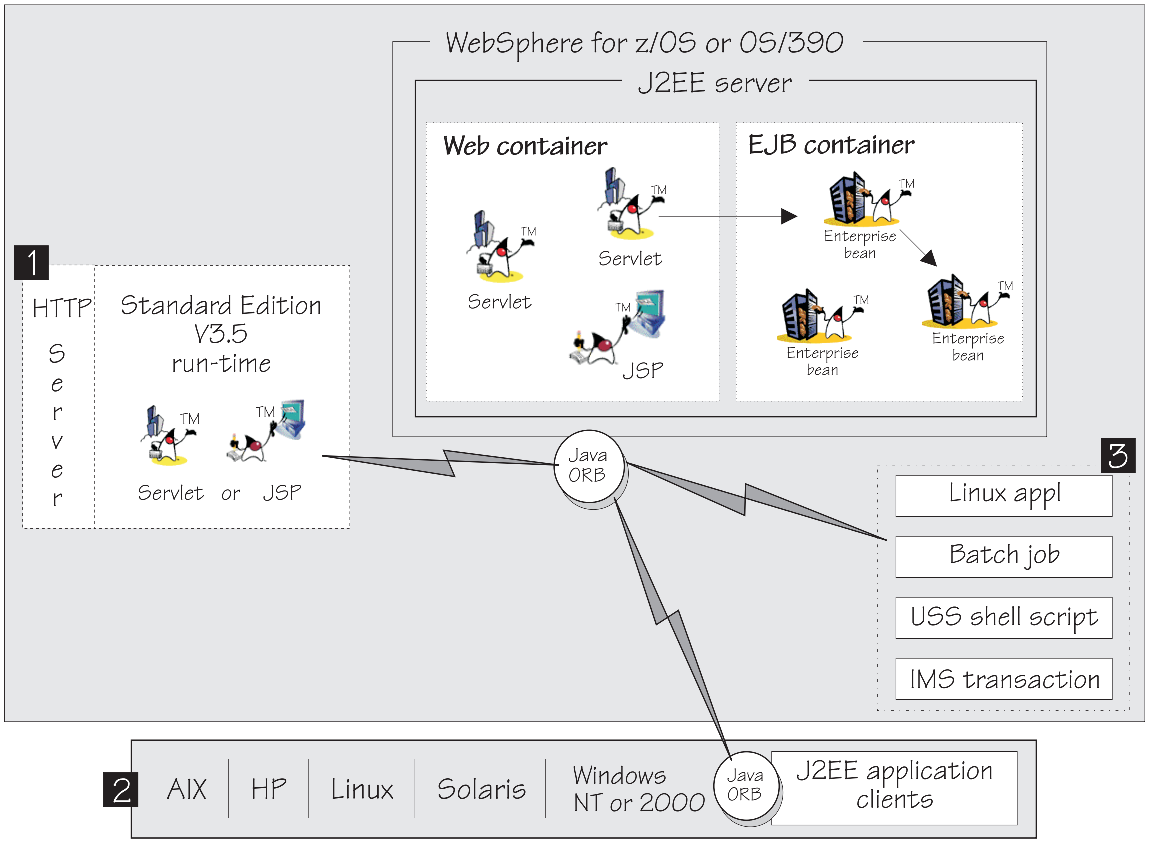


Figure 1. Potential clients of application components installed in the J2EE server

1. Web components, also known as Web applications, can run in the following environments:
   - The WebSphere for z/OS J2EE server's Web container.

- The WebSphere for z/OS V3.5 run-time environment (hereafter referred to as the V3.5 run-time).

  Web applications that currently run in the WebSphere Application Server Standard Edition V3.5 environment and are J2EE compliant can be migrated to run in the WebSphere for z/OS Web container.

2. J2EE application clients that run on platforms other than z/OS and OS/390 are also potential clients. These applications are considered remote clients because they do not reside on the same z/OS or OS/390 image as the Enterprise beans they use.

   The supported platforms— Windows NT or 2000, AIX, HP-UX, Linux, Netware, OS/2, OS/400, and Solaris— are also known as distributed, or workstation, platforms. WebSphere products run on these platforms, but are not necessarily required for remote J2EE application clients to access J2EE application components in a WebSphere for z/OS J2EE server.
3. Another set of potential clients is native Java applications, which are programs that run in the z/OS UNIX System Services environment, or in other subsystems, on either z/OS or OS/390. These applications can be either local or remote clients, depending on whether or not they reside on the same z/OS or OS/390 image as the Enterprise beans they use.

Enterprise beans and Web applications that run in a WebSphere for z/OS J2EE server may also be clients of application components in another WebSphere for z/OS J2EE server. These clients can be either local or remote clients, depending on whether or not they reside on the same z/OS or OS/390 image as the Enterprise beans they use.

Figure 2 on page 6 illustrates the components that comprise the WebSphere for z/OS run-time environment, including the z/OS or OS/390 functions that WebSphere for z/OS uses to provide a robust, reliable, and scalable environment for J2EE applications. This diagram represents how the WebSphere for z/OS product is initially configured in a monoplex on z/OS or OS/390. The names of individual WebSphere for z/OS components match the names your installation's system programmers are instructed to use when first installing the WebSphere for z/OS product.

The run-time servers are labelled with the term "server instance". In WebSphere for z/OS, the functional component on which applications run is called a server instance. Server instances comprise address spaces that actually run code.

A "server", on the other hand, is a logical grouping of replicated server instances. Servers allow you to partition workloads into separate server instances, but still refer to them as a single unit. This is particularly important in sysplex environments, where each system in the sysplex might be running a replicated server instance, but clients outside the sysplex address them as a single server. The client does not know which server instance is actually doing work on its behalf; in fact, a subsequent work request from the client may, due to workload balancing, be served by a different server instance in the sysplex.

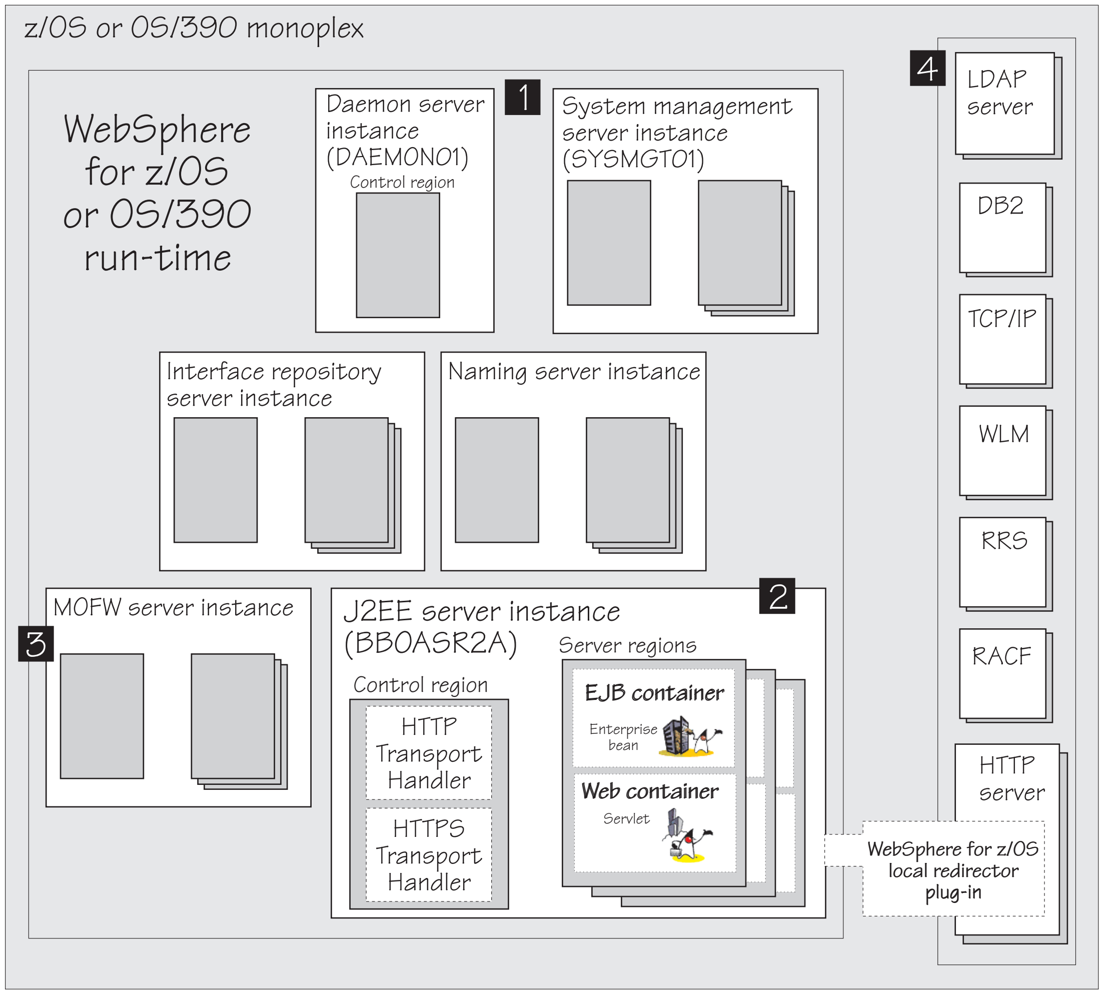


Figure 2. The WebSphere for z/OS server configuration in a monoplex

The following numbered items correspond to the numbers in Figure 2:

1. A full WebSphere for z/OS run-time environment includes several servers: the Daemon, System Management, Naming, and Interface Repository server instances. Though not directly part of WebSphere for z/OS, the run time requires a Lightweight Directory Access Protocol (LDAP) server.
2. WebSphere for z/OS provides a J2EE server instance for J2EE application components: Enterprise beans and Web applications. J2EE servers contain at least one EJB container and one Web container. The EJB container manages Enterprise beans, while the Web container manages Web applications (servlets and JavaServer Pages). The WebSphere for z/OS run time also includes a V3.5 run-time that runs in the HTTP server address space.

   Clients can access Web applications in the following ways:

   - By configuring an HTTP or HTTPS Transport Handler for each J2EE server to receive HTTP or HTTPS requests directly into the control region for that server. Using this option removes the requirement to have an IBM HTTP Server for z/OS configured with the Local Redirector plug-in to route requests to the Web container. IBM recommends using the HTTP/HTTPS Transport Handlers if their capabilities meet your needs.
   - By using the IBM HTTP Server for z/OS in conjunction with the Local Redirector plug-in. Using this transport allows the IBM HTTP Server for

z/OS to serve as the entry point for HTTP requests into the Sysplex. The plug-in routine then forwards the requests to the Web container for processing.

3. WebSphere for z/OS also provides a Managed Object Framework (MOFW) server instance, which provides a run-time environment for CORBA-compliant components.

   Notes:

   a. Servlets can drive CORBA-based Java business objects in a MOFW server.
   b. Enterprise beans and Java business objects can interoperate.
   c. C++ business objects in a MOFW server, however, cannot access application components in a J2EE server.

4. The run-time server instances use other z/OS or OS/390 functions, such as z/OS UNIX, and TCP/IP. Part of installing WebSphere for z/OS includes configuring these functions for use by the run-time. Details about installing, customizing, and configuring the WebSphere for z/OS run-time components appears in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834.

Chapter 2. Overview of application development and tools

Because Enterprise beans and Web applications must conform to J2EE packaging standards, WebSphere for z/OS supplies tools to help you prepare J2EE applications for installation in a WebSphere for z/OS J2EE server. The following topics briefly review how to:

- Create, assemble and install (or deploy) J2EE applications in a WebSphere for z/OS J2EE server. The assembly stage requires the use of WebSphere Studio Application Developer and 390fy, which is the preferred method of deploying the applications, or the WebSphere for z/OS Application Assembly tool, which is discussed in this chapter.
- Define and activate a WebSphere for z/OS J2EE server. This process requires the use of the WebSphere for z/OS Administration application. The Administration application is also known as the Systems Management End-User Interface (SM EUI).

  If you have used the WebSphere Enterprise Edition for OS/390 product, you will notice some differences when you use the Administration application. See Chapter 3, "Overview of J2EE server definition and activation," on page 13.

To understand the process description in this chapter, you might need to review some J2EE terminology: a Java 2 Enterprise Edition (J2EE) application is comprised of J2EE modules, which, in turn, are comprised of J2EE components:
- Enterprise beans
- Web components; that is, servlets or JavaServer Pages (JSPs)
- Application clients
- Applets

A J2EE module may contain one or more of the same type of component. J2EE modules are archives: Java archive (JAR) files or Web application archive (WAR) files. J2EE applications, collections of J2EE modules, are packaged in Enterprise archive (EAR) files. You can install J2EE applications in WebSphere for z/OS only when they are packaged in EAR files.

The J2EE application components that WebSphere for z/OS currently supports in its J2EE server are Enterprise beans and Web components. The J2EE server supports Enterprise beans through its EJB container, and supports servlets and JSPs through its Web container.

To create the supported components for a J2EE application, you need to be familiar with the Sun Microsystems specifications for each type of component— Enterprise bean, servlet, or JSP— and the specification levels that WebSphere for z/OS supports. With that knowledge, you may begin the development and deployment process, as illustrated in Figure 3 on page 10:

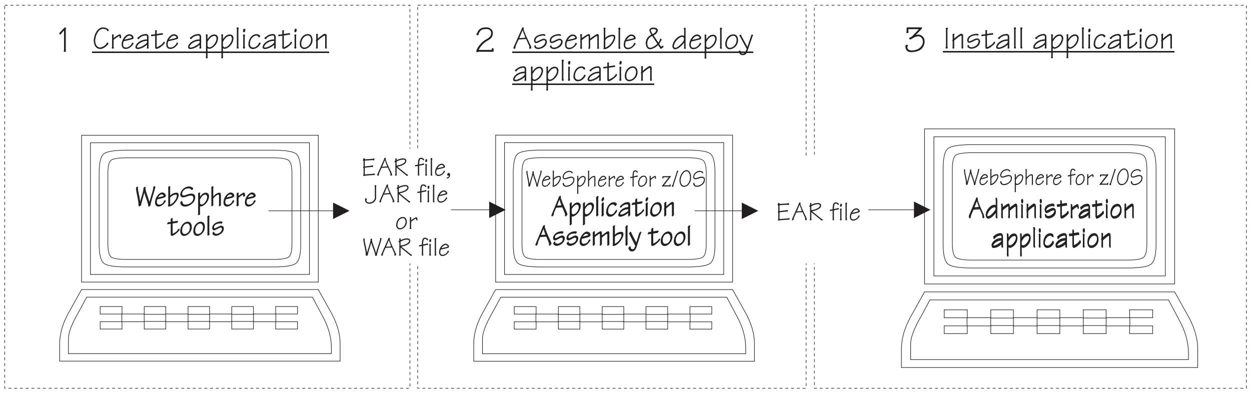

1 Create application
2 Assemble & deploy application
3 Install application
WebSphere tools
EAR file, JAR file or WAR file
WebSphere for z/OS Application Assembly tool
EAR file
WebSphere for z/OS Administration application

Figure 3. Tools and output for developing, assembling, and installing components in a J2EE server

1. According to your business goals, define and implement application components and the associated classes or files that each component requires. To develop application components, you may use a tool like IBM's WebSphere Studio Application Developer or Application Developer Integration Edition, orVisualAge for Java.

   As part of the component development process, the tools you use create a deployment descriptor, which contains attribute and environment settings that you select to define how a J2EE server is to manage each application component's lifecycle and resources. You may test these definitions in the workstation development environment, because they are platform-independent specifications.

   When you have completed this stage of the process, you have one or more of the following artifacts, as illustrated in Figure 4 on page 11:
   - Enterprise beans, their classes and deployment descriptor packaged in Java archive (JAR) files or Enterprise archive (EAR) files
   - Web applications, consisting of any combination of servlets, their classes and descriptors, JSPs, or static files, such as HTML or GIFs, packaged in Web application archive (WAR) files or EAR files

   Notes:

   a. Each JAR file may contain one or more Enterprise beans; similarly, each WAR file may contain multiple servlets or JSPs. EAR files may contain one or more Enterprise beans, one or more Web applications, or both Enterprise beans and Web application components. These files become input for the next stage of the process: assembly and deployment.
   b. Initial JAR, WAR and EAR file assembly can be accomplished using WebSphere development tools; these files then can be further decorated and deployed using a WebSphere for z/OS application assembly tool. "A WebSphere application assembly tool" means either the z/OS edition of the Application Assembly tool, the WebSphere Studio Application Developer tool, the WebSphere Studio Application Developer Integration Edition tool, or Direct Deployment Tool/390fy.
   c. Java AWT classes are supported but an application containing a Java AWT class must be hosted in the Local Redirector Plug-in. If you want to use Java AWT classes, you must make the following changes to the HTTP Server's httpd.envvars file:
      - Set the DISPLAY environment variable to **x.xx.xx.xx:0.0**, where **x.xx.xx.xx:0.0** is the XWindows environment variable stating the IP address of the XTerminal.

- Set the LANG and LC_ALL environment variables to **C**.
- Add **/usr/lib** to the LIBPATH. This enables you to pick up Xm.dll and X11.dll.

More information about using AWT classes is available at URL:
`http://java.sun.com/products/jdk/awt/`

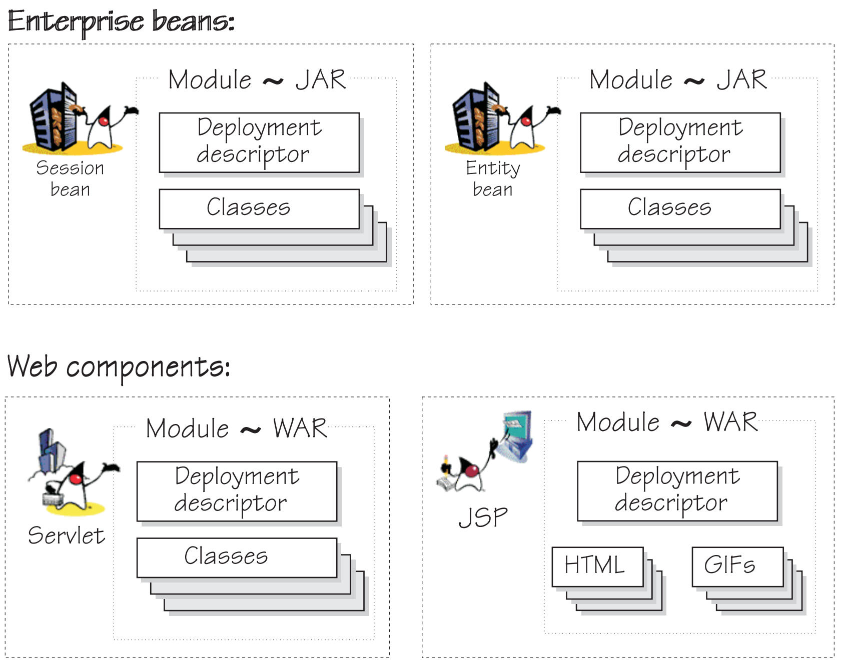


Figure 4. Supported J2EE application components

2. Assemble and deploy J2EE application components

   During application assembly and deployment, you need to assemble these modules into an Enterprise archive (EAR) file, which is the only type of file that WebSphere for z/OS accepts for installation in a J2EE server.

   The preferred method of deploying applications is to use WebSphere Studio Application Developer and Direct Deployment Tool/390fy, but you may use the WebSphere for z/OS Application Assembly tool instead.

   During the assembly and deployment process, these tools:
   - Generate code for z/OS or OS/390, including remote interfaces, home interfaces, ties and stubs, keys, handles, finder helpers, and code related to persistence.
   - Convert the deployment descriptors for V1.0 Enterprise beans to match the V1.1 specification level. This capability enables WebSphere for z/OS to support V1.0 beans.
3. Install the J2EE application

   The preferred method of installing applications is to use Direct Deployment Tool/390fy, but you may use the WebSphere for z/OS Administration application instead.

   Through the installation process, deployment descriptors are customized for the WebSphere for z/OS environment, and application components are loaded into the WebSphere for z/OS J2EE server. Specifically:
   - Application artifacts (including EAR, JAR and WAR files) are transferred from the workstation to z/OS or OS/390.

- Application metadata is stored so that WebSphere for z/OS can access and manage J2EE application components.
- Bean and servlet resources and references are resolved through the use of the Java Naming and Directory Interface (JNDI).
- The container parameters for the J2EE server are configured through the deployment descriptors for installed applications, modules, components, and methods.
- Web applications are provided with a fully qualified URI that enables the WAR files and the EJB JAR files to be accessed through HTTP protocol when requested by a client.

Chapter 3. Overview of J2EE server definition and activation

Once you have developed and assembled an executable J2EE application, you need to install the application in an appropriate run-time environment. For the z/OS or OS/390 platform, the run-time environment is a WebSphere for z/OS application server (J2EE server). Depending on your installation’s policies, a J2EE server might be already available for you to use for testing applications. If not, you need to create a new J2EE server. Creating a new server involves completing a combination of tasks by hand or through the WebSphere for z/OS Administration application. The Administration application is also known as the Systems Management End-User Interface (SM EUI).

Because the manual tasks require some expertise with the z/OS or OS/390 operating system, each task is typically performed by a system programmer, database administrator, or security administrator. IBM provides samples that can help you complete these tasks, even if you are not very familiar with z/OS or OS/390, but you should consult with experts at your installation, if necessary.

The tasks you accomplish through the Administration application do not necessarily require familiarity with z/OS or OS/390. You might, however, find the tasks easier if you understand that you start by defining a model run-time environment for your application, and complete the process by activating the model into a working z/OS or OS/390 application.

This chapter provides an overview of how you gradually build this server model and turn it into a run-time environment that exploits traditional strengths of the z/OS or OS/390 operating system. To accomplish these tasks, you must make sure the Administration application has a connection to a properly configured and active WebSphere for z/OS installation. Detailed instructions for creating a server and installing your applications appear in Part 2, “Creating, assembling and deploying J2EE server applications,” on page 107.

1. Define the J2EE server, J2EE server instance, and J2EE resources

   When you use the Administration application to define a J2EE server, you create a model that includes the elements illustrated in Figure 5 on page 14.

© Copyright IBM Corp. 2000, 2003

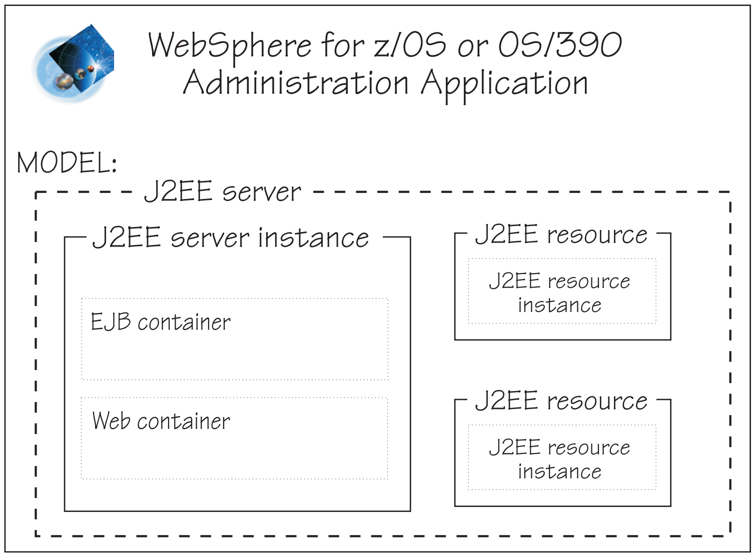


Figure 5. Model of a J2EE server

The model includes additional elements, such as system and sysplex, that define how the J2EE server fits into a z/OS or OS/390 configuration for test or production. For the purpose of defining a run-time environment for your application, however, you may concentrate on these model elements:

- A server that represents the application environment. A server is an entity that is responsible for a certain type of work that runs on z/OS or OS/390.

  A server also is a logical grouping of replicated server instances. Servers allow you to partition workloads into separate server instances, but still refer to them as a single unit. Servers are generic to a sysplex.
- A server instance in which your application will run.

  A server instance is a collection of address spaces that work together to receive requests for your application's components, and to run and manage your application's code. The server instance is responsible for running and managing your application components through an EJB container for Enterprise beans, or a Web container for servlets and JSPs. Server instances are specific to only one z/OS or OS/390 system.
- A J2EE resource and J2EE resource instance. They represent, respectively, generic types of system resources and the specific subsystems that manage those types. For example, DB2 is a type of z/OS or OS/390 system resource, and a specific DB2 subsystem might manage all of the DB2 databases and tables on that system.

2. Install Enterprise archive (EAR) files containing your J2EE applications

   Your J2EE applications also become part of the server model. As part of the installation process:
   - Application artifacts (including EAR, JAR and WAR files) are transferred from the workstation to z/OS or OS/390.
   - Application metadata is stored so that WebSphere for z/OS can access and manage J2EE application components.
   - Bean and servlet resources and references are resolved through the use of the Java Naming and Directory Interface (JNDI).
   - The container parameters for the J2EE server are configured through the deployment descriptors for installed applications, modules, components, and methods.

- Web applications are provided with a fully qualified URI that enables the WAR files and the EJB JAR files to be accessed through HTTP protocol when requested by a client.

Figure 6 illustrates the result of the server definition and application installation process: a model of the application server instance with:

- An EJB container (for Enterprise beans), and possibly a Web container (for servlets and JSPs).
- Connections to the J2EE resources, which WebSphere for z/OS creates as part of the application installation

Note: Currently, if you are installing a J2EE application containing servlets or JSPs, you must complete some additional tasks to configure a Web container for the J2EE server. Then you may use the Administration application to install your application. Details appear later in this book, in the step-by-step instructions for creating a J2EE server.

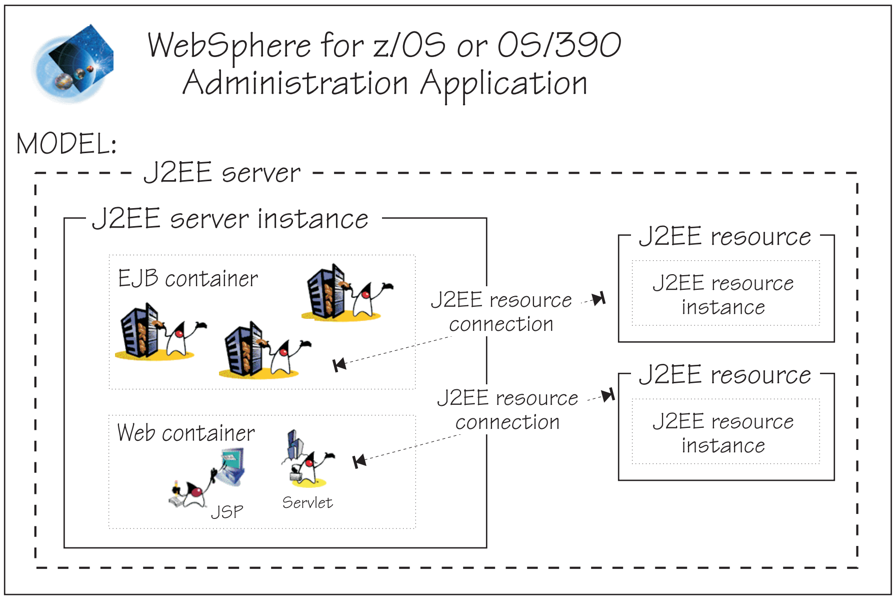


Figure 6. Installing a J2EE application in a J2EE server

3. Convert the conversation model into an active J2EE server run-time environment

   Once you have a model of the run-time environment for your sample application, you start the last phase of this process: converting the model into an active run-time environment on z/OS or OS/390.

   First, you use the Administration application to commit the server model, which is analogous to permanently saving the definition.

   Then you use the Administration application to activate the server. Figure 7 on page 16 illustrates how the model elements correspond to active elements in the z/OS or OS/390 system.

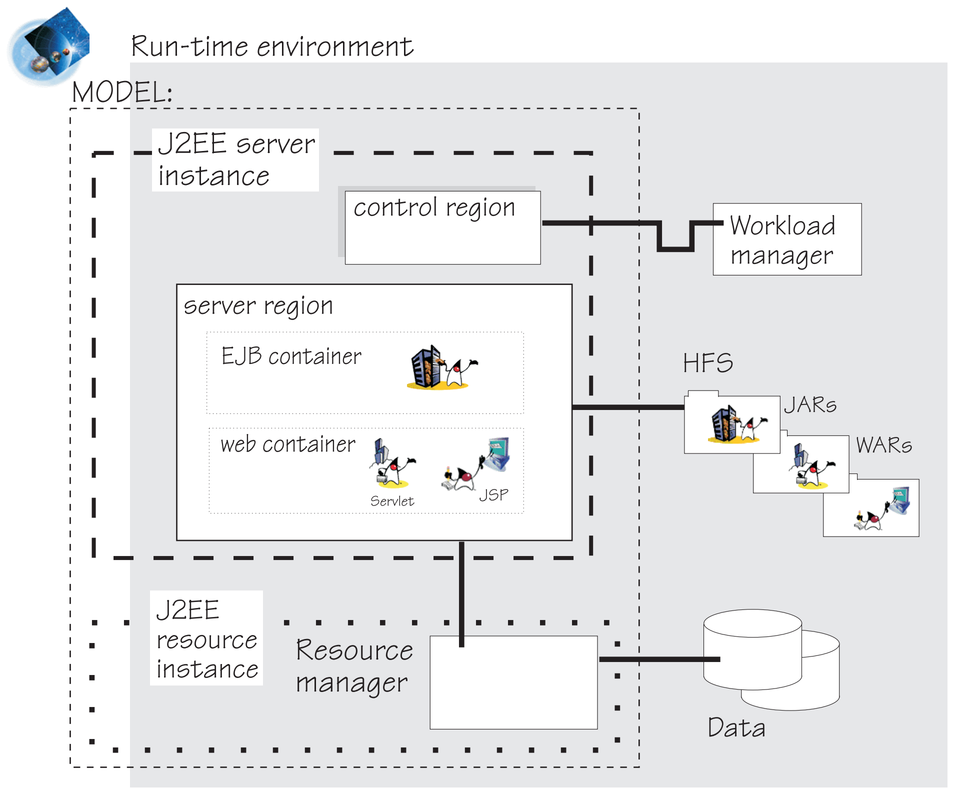


Figure 7. An active J2EE application server

In particular, note the following elements:

- A server instance consists of:
  - A control region that receives and queues client requests to the z/OS or OS/390 workload manager (WLM).
  - One or more server regions (z/OS or OS/390 address spaces). A server region consists of several functions that work together to run and manage your application's code. A Java virtual machine (JVM) runs in a server region address space; your application components will run in this JVM.

    WLM starts additional server regions depending on the volume of incoming requests.
- The server region containers manage the lifecycle of application components installed in this server. Each server region can find the executable application code that was installed in the HFS.
- A J2EE resource instance is equivalent to one subsystem managing one type of resource. Each server instance may be connected to one or more J2EE resource types and subsystems.

After you activate a server model, you can start a J2EE server instance that is ready to handle client requests. The sequence of events happens like this:

1. The control region receives client requests through TCP/IP, and sends them to WLM.
2. WLM balances the workload of client requests among server regions, starting additional server regions as necessary.
3. When a server region receives a client request to process, it determines whether the request is for an Enterprise bean or a Web component, and directs the

request to either the EJB container or Web container. The container then locates the code for the referenced application component, and processing for the client request begins.

4. If the use of J2EE resource is necessary to complete processing on behalf of the client, the server region connects to an actual z/OS or OS/390 subsystem, such as DB2. The server region may connect to any subsystem that is defined to the server instance as a J2EE resource.

Chapter 4. A closer look at the J2EE server

The following table lists the topics in this chapter, which describe how WebSphere for z/OS satisfies or implements Java 2 Platform, Enterprise Edition (J2EE) requirements. Reading these topics can help you understand how to exploit the WebSphere for z/OS environment so that your J2EE applications benefit from traditional z/OS and OS/390 security, scalability, and reliability.

Understanding...	**Associated information (See . . .)**
WebSphere for z/OS implementations	• "Security" • "Naming" on page 42 • "Application programming interfaces" on page 43 • "Connectors" on page 64
How to exploit the WebSphere for z/OS environment and services	• "The WebSphere for z/OS environment for Web applications" on page 80 • "Web services" on page 105

Security

WebSphere for z/OS provides a variety of security services that allow you to control access to application components running in a J2EE server and, in turn, the z/OS or OS/390 system resources that those components use. These security services address client and server authentication in a distributed network, and authorization to use specific resources. At your installation, system programmers and security administrators have already selected which security mechanisms are appropriate for systems, based on their knowledge of the installation and its resources. While you might not be directly involved in setting up security, knowing which mechanisms are in use can help you determine what additional controls, if any, are required for new application components that you plan to assemble, deploy, and install in a WebSphere for z/OS J2EE server.

For example, if you plan to deploy an application that does not have any security checking in its logic, you might need to know how you can use z/OS or OS/390 security mechanisms to protect your application's methods and data, to authenticate clients, to propagate client identities, or to encrypt confidential data that might flow on the network.

The following topics summarize WebSphere for z/OS support for security:

- Network and system security mechanisms for authentication and authorization, which are discussed in detail in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834, and
- Security-related application programming interfaces and definitions for controlling access to Enterprise beans or specific bean methods.
- Security-related application programming interfaces and definitions for controlling access to Web application components.

Authentication services for J2EE clients and servers

Proper security for any system requires that users or programs prove their identity; that is, authenticate themselves. Figure 8 on page 20 lists the user identification

© Copyright IBM Corp. 2000, 2003

and authentication services that are available for identifying J2EE clients, depending on where they reside in relation to the J2EE application components they want to use. Authentication mechanisms are negotiated separately for each combination of client and server.

Keep in mind that application components in a WebSphere for z/OS J2EE server also can be clients of application components in other WebSphere for z/OS J2EE servers. In other words, clients might request a service that requires a server to forward the request to another server. In such cases, the system must handle delegation, the availability of the client identity for use by intermediate servers and target servers.

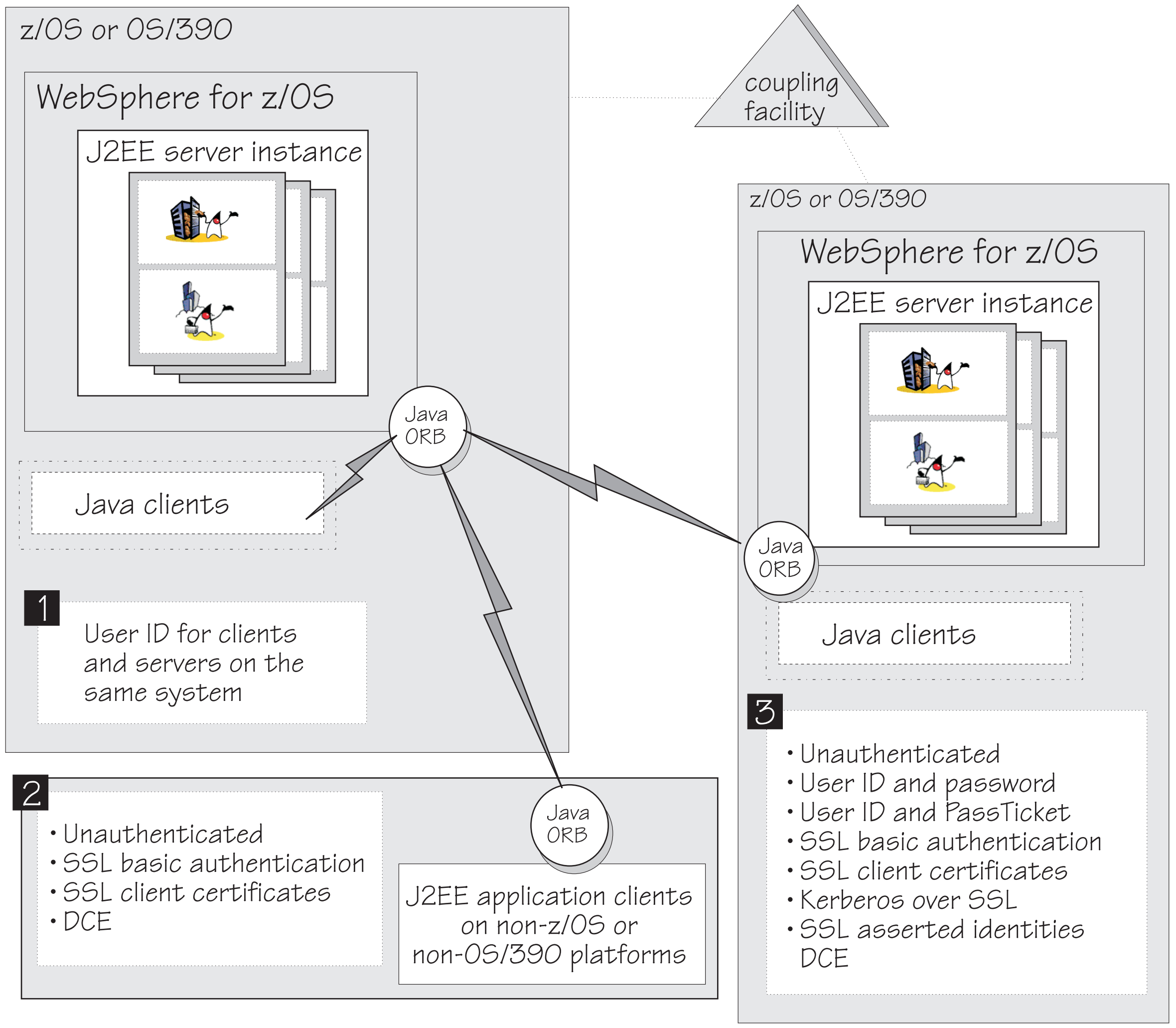


Figure 8. Client identification and authentication

The following list corresponds with the numbered items in Figure 8:

1. Local clients and servers, which run in the same z/OS or OS/390 system, use their user IDs to identify themselves when requesting a service. Just like other z/OS or OS/390 applications, WebSphere for z/OS uses the operating system to keep track of the user identities and makes calls to the security service during the execution of a piece of work.

2. Clients and servers that run on non-z/OS or non-OS/390 platforms can be authenticated using several security mechanisms:
   - Configure J2EE servers to accept unauthenticated clients, which provides no protection from tampering. In this case, the server establishes a default identity for every unauthenticated request received.
   - Use Secure Sockets Layer (SSL) security, which provides encryption and authentication services. Encryption ensures the integrity of messages in a network.

     WebSphere for z/OS security provides several types of client authentication that take advantage of SSL:
     - Basic authentication, in which the server proves its identity by passing a digital certificate to the client, which passes back an encrypted user ID and password known by the server.
     - Client certificate support, in which both the server and client supply digital certificates to prove their identities to each other.
   - Use Distributed Computing Environment (DCE), which uses a third-party verification technique that verifies that clients are communicating with the correct servers, and servers are communicating with the correct clients. DCE also allows you to encrypt messages and check for tampering.
3. Clients and servers that run on other z/OS or OS/390 systems in the same sysplex can be authenticated using any of the mechanisms for network cases listed in the previous item (item 2). All network protocols are supported between clients and servers within the sysplex.

   Clients and servers in the same sysplex also may use the following mechanisms:
   - PassTickets, in which the client's user ID is used for identification and a PassTicket for authentication. A PassTicket is a one-time-use password that is dynamically generated.
   - Kerberos over SSL, in which Kerberos client authentication is used together with SSL to provide a complete authentication mechanism. SSL serves as the transport layer, providing message encryption and initial authentication, while Kerberos provides the ability to securely exchange data between client and server.
   - SSL identity assertion, or trusted association, in which an intermediate server sends already verified client identities to a target server. Once the target server recognizes the intermediate server as trusted, the target server can avoid the process of authenticating client requests received from the intermediate server.

Note: Only the following mechanisms allow you to propagate client identity in server-to-server interactions:
- Within the same z/OS or OS/390 system: WebSphere for z/OS uses the operating system's control data to propagate client identity.
- Within a sysplex:
  - Passtickets
  - Kerberos over SSL
  - SSL identity assertion

Authorization controls for J2EE clients and servers

In addition to supporting various authentication mechanisms, WebSphere for z/OS also supports various authorization controls to prevent inadvertant or malicious destruction of resources. Once your installation has determined how to authenticate clients and servers in a distributed environment, security administrators need to determine whether these clients and servers require authorization to use specific WebSphere for z/OS, z/OS, or OS/390 resources. When a request flows from a client to a server, or from a server to another server, WebSphere for z/OS may check the user ID associated with a request to determine whether this user has the authority to make such a request.

WebSphere for z/OS requires a subset of possible authority checks, and the authorization mechanisms for those checks are set up when your installation's system programmers customize WebSphere for z/OS, before they can run the installation verification programs (IVPs). Depending on your installation's security policies, you might have to work with a security administrator to modify these authorization mechanisms, or set up additional mechanisms tailored to the processing of the J2EE applications you plan to install.

Figure 9 illustrates the system resources that you might need to protect through authorization controls. The characteristics of the J2EE application components you are using, along with your installation's security policies, determine which controls are required.

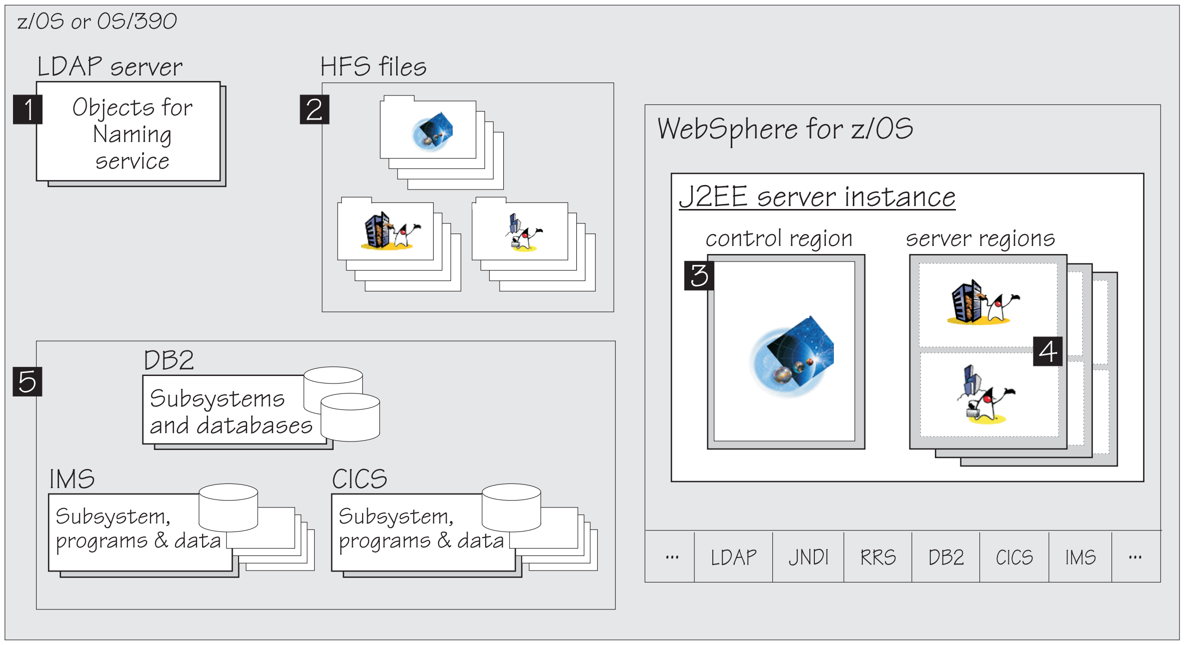


Figure 9. Authorization controls in the WebSphere for z/OS environment

The following list corresponds with the numbered items in Figure 9, and highlights the authorization mechanism that your installation uses to protect each resource:

1. For WebSphere for z/OS, the LDAP component of the z/OS or OS/390 Security Server provides the directory services for the Java Naming and Directory Interface (JNDI) naming and interface repository services. The contents of the directory are stored in DB2 tables. Your installation can set up LDAP access control lists (ACLs) to protect its objects.
2. WebSphere for z/OS uses specific directories in the hierarchical file system (HFS) for configuration data. During installation and customization, your system programmers check to make sure the appropriate run-time servers are assigned ownership of or permission to access specific files.
3. Each WebSphere for z/OS J2EE server is comprised of one control region and one or more server regions. Control regions run only authorized programs that

are loaded from Authorized Program Facility (APF) libraries. Server regions, on the other hand, contain application code that usually runs unauthorized. Because of this fact, all server regions require authorization to profiles in the SERVER class, which controls access to authorized routines in the control region.

Additionally, your installation can use the CBIND class to restrict a client's ability to access servers, or may deactivate the class if this kind of access control is not required.

Note: On z/OS or OS/390, each control region, server region, and client has a user ID associated with it.

4. J2EE application components can be deployed to set identities that:
   - Control access to individual components or their methods.

     Application developers or assemblers can deploy J2EE application components to use security roles, which identity callers with authorization to invoke the component or its methods. For security roles to work, the deployer or z/OS or OS/390 security administrator must define the security roles to the SecureWay Security Server for z/OS and OS/390 (RACF) in the EJBROLE class.
   - Control access to any J2EE resources or z/OS or OS/390 system resources, during execution of a J2EE application component or individual method.

     By default, during the execution of an Enterprise bean or a Web application, WebSphere for z/OS uses:
     - Caller identity on downstream processing requests for J2EE resources (such as other beans).
     - Server identity on downstream processing requests for z/OS or OS/390 system resources or resource managers (such as DB2).

     Application assemblers or deployers can change this default behavior by deploying a J2EE application component or its methods with RunAs or synchronization settings. Depending on the values used for these settings, the deployer or z/OS or OS/390 security administrator might need to define RACF profiles for the EJBROLE class, and the deployer also might need to enable synchronization in the WebSphere for z/OS J2EE server.
5. Resource managers such as DB2, CICS, and IMS have implemented their own resource controls to protect subsystems, programs, and data. If your installation uses any of these controls, the appropriate profiles must be defined in the security product in use on z/OS or OS/390.

Table 2 on page 24 provides a summary of all authorization controls, and indicates whether they are required or optional.

Table 2. Alphabetical summary of authorization controls for the WebSphere for z/OS environment

Method of access control	Protected resource	Required/optional for J2EE server	Required/optional for J2EE client[1]
CBIND class[2]	Through profile CB.BIND.*srvname*: WebSphere for z/OS J2EE servers	Optional. If your installation uses this CBIND class profile, the WebSphere for z/OS Systems Management user IDs require read access to this profile for each new J2EE server you create for your applications.	Optional. If your installation uses this CBIND class profile, J2EE clients require access to the J2EE servers that contain application components that the client will use.
	Through profile CB.*srvname*: J2EE application components installed in the server	Optional. If your installation uses this CBIND class profile, the WebSphere for z/OS Systems Management user IDs require read access to this profile for each new J2EE server you create for your applications.	Optional. If your installation uses this CBIND class profile, J2EE clients require access to the J2EE servers that contain application components that the client will use.
DB2 GRANT statements **Alternative:** Use the DSNR class.	DB2 databases and tables	Optional. If your installation is using DB2 access controls for DB2 databases, grant access for all control regions and server regions.	Optional. Use only for J2EE application components or connectors that use an operating system identity other than the J2EE server identity.
DB2 secondary authorization IDs	DB2 databases and tables	Optional. If applications installed in the J2EE server contain container-managed (CMP) and bean-managed (BMP) Enterprise beans that use unqualified table references, define an SQL ID for the DB2 datasource associated with the server. For further information about using SQL IDs for datasources, see "Overview of SQLID for managed datasources" on page 34.	Not applicable.
DSNR class **Alternative:** Use DB2 GRANT statements.	DB2 subsystem	Optional. If your installation is using DB2 access controls for DB2 databases, grant access for all control regions and server regions.	Optional. Use only for J2EE application components or connectors that use an operating system identity other than the J2EE server identity.
EJBROLE or **GEJBROLE** class[3] If J2EE applications using security roles are installed, EJBROLE class is required; GEJBROLE is optional.	J2EE application components installed in a J2EE server	Optional. If J2EE applications using method permissions are deployed in a J2EE server with the default RunAs server setting, either: • Permit the server region identity to the defined security roles, or • Redeploy the application with a new security role defined for server use.	If J2EE applications using security roles are deployed in a J2EE server with the RunAs caller or role setting, permit the J2EE client or role identity to the EJBROLE class.
FACILITY class (IMSXCF .OTMACI)	IMS subsystem and resources	Optional. Use only for J2EE servers that delegate client requests to IMS through OTMA, passing the server identity.	Optional. Use only for J2EE application components or connectors that use an operating system identity other than the J2EE server identity.

Table 2. Alphabetical summary of authorization controls for the WebSphere for z/OS environment (continued)

Method of access control	**Protected resource**	**Required/optional for J2EE server**	**Required/optional for J2EE client**[1]
HFS file permissions	HFS files	Required. Specific permissions are set for J2EE server configuration files; see *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834 for further details.	Required only if the J2EE server property `Enable Setting OS Thread Identity to RunAs Identity` is set, and the RunAs identity is caller or security role.
LDAP server access control lists (ACLs)	LDAP objects (related to Naming services)	Optional. Typically, installations use a general ANYBODY user ID with read access to the LDAP name space, which allows both servers and clients to access naming services. If necessary, see *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834 for further details.	Not applicable.
SERVER class	WebSphere for z/OS control regions	Required. Server regions must have read access to profiles in the SERVER class.	Not applicable.
SURROGAT class (*.DFHEXCI)	CICS subsystem and resources	Optional. Use only for J2EE servers that delegate client requests to CICS through EXCI, passing the server identity.	Optional. Use only for J2EE application components or connectors that use an operating system identity other than the J2EE server identity.
J2EE server property `Enable Setting OS Thread Identity to RunAs Identity`	The operating system identity on the thread of execution	Optional. Use only if you want to enable J2EE application components or connectors to change the execution identity for non-J2EE resources.	Not applicable.

1. This column applies to the J2EE identity that might request access to a protected resource. By default, the J2EE identity is that of the client initiating the request. Because this identity is modifiable by RunAs settings, however, the J2EE identity might be the J2EE server region ID or the user ID associated with a security role.
2. These two profiles use the variable *srvname* which is the name of the J2EE server.
3. For further details about using security roles, security identities, and the EJBROLE and GEJBROLE classes, see "Authorization controls for J2EE application components."

In other topics in this book, step-by-step procedures for setting up J2EE servers and clients list which security controls might need to be in place. Most of these authorization controls are discussed in more detail in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834.

Authorization controls for J2EE application components

In addition to supporting authorization controls to protect J2EE servers and the application components that run in them, WebSphere for z/OS also supports two distinct forms of authorization controls:

- Declarative forms of authorization, in which application assemblers specify access to a J2EE application component at the method level, and
- Application programming interfaces (APIs), which application programmers or bean providers use in the logic of their components.

These controls provide authorization checking on a more granular level than client-to-server interaction.

Suppose your installation uses a banking application containing an Enterprise bean that manages all activities for individual bank accounts. Your installation might want, for example, to allow all bank tellers to invoke this bean for deposits and withdrawals, but to allow only bank managers to use this bean for closing accounts. In this case, all bank tellers and managers require access to the Enterprise bean, but only managers should be authorized to use the bean's `closeAccount` method. Your installation can accomplish these authorization checks using security roles and method permissions.

These authorization checks are part of the security management model defined in the Sun Microsystems J2EE and EJB specifications. Using this security model:

1. The bean provider (also known as the application developer) codes beans that are intended to be portable across all J2EE platforms. With this goal in mind, bean providers might not suggest any security policies for individual J2EE application components, but they have the option of doing so by defining security-role-reference elements in a component's deployment descriptor. These elements identify types of users that should have authority to invoke a component or its methods. Bean providers also can use security application programming interfaces (APIs) to perform authorization checks on a component or method level. These APIs are:
   - The `getCallerPrincipal` method, which allows bean methods to obtain the current caller's security context.
   - The `isCallerInRole` method, which allows bean providers to code security checks that cannot be achieved through the use of method permissions defined in a deployment descriptor.

   These interfaces are platform-independent; knowledge of the component's potential run-time environments is not required to use these mechanisms.
2. Application assemblers combine Enterprise beans and other components into J2EE applications that address a business problem or implement a business process. These components may come from diverse sources, so part of the application assembler's job is to integrate these components so that they fit into the customer installation's security implementation. To do so, application assemblers define a security view for J2EE application components, which consists of:
   - Security roles, which denote logical permissions or general types of users, rather than actual users and user groups in a security implementation. Security roles may be defined at the bean or application level. If the bean provider used any security role references, which are defined at the method level, the application assembler must link those references to one of the security roles.
   - Method permissions, which indicate the security role that may invoke a specific bean home or remote interface method.

   This logical view maps to the actual security implementation on a particular platform. Knowledge of security policies— not security implementation— is required to define this application security view. Application assemblers and deployers work with security administrators to understand installation security policies.

   Application assemblers must use a WebSphere application assembly tool to define the security view. "A WebSphere application assembly tool" means either the z/OS edition of the Application Assembly tool, the WebSphere Studio Application Developer tool, the WebSphere Studio Application Developer Integration Edition tool, or Direct Deployment Tool/390fy.

3. Security administrators work with application assemblers and deployers to set or communicate installation security policies. If adjustments to the installation's security implementation are required for particular J2EE applications, the security administrator completes this work.

   To support application security controls in the WebSphere for z/OS environment, security administrators need to define or update RACF profiles for the EJBROLE class. The EJBROLE class is required for applications that use security roles for method permissions, or use the `isCallerInRole` method. The GEJBROLE class, a grouping class, is also an option that might decrease the number of required RACF definitions to support applications that use security roles or method permissions.

For further details, read "Security roles and method permissions," which uses the banking application example to illustrate how to set up and use security roles to control access on a method level.

Security roles and method permissions

To understand how security roles and method permissions work in the WebSphere for z/OS environment, consider the procedure illustrated in Figure 10 on page 28, which highlights key steps for deploying an `Account` Enterprise bean in a WebSphere for z/OS J2EE server. Through this example, the list following the diagram explains the required authorization controls and process for deploying an Enterprise bean that uses programmatic security checks. To make these security checks function properly during bean execution, the WebSphere for z/OS security domain (RACF) must have user profiles defined for the EJBROLE class.

Assume that the sample Enterprise bean is one component of a banking application, and that the bean itself is designed to manage activities for a banking account, such as withdrawals, deposits, and so on. Specifically, the remote interface includes `deposit`, `withdraw`, `getBalance`, `setBalance` and `closeAccount` methods. The home interface includes a number of methods, such as `findByPrimaryKey`, for creating, removing and finding instances of the `Account` bean.

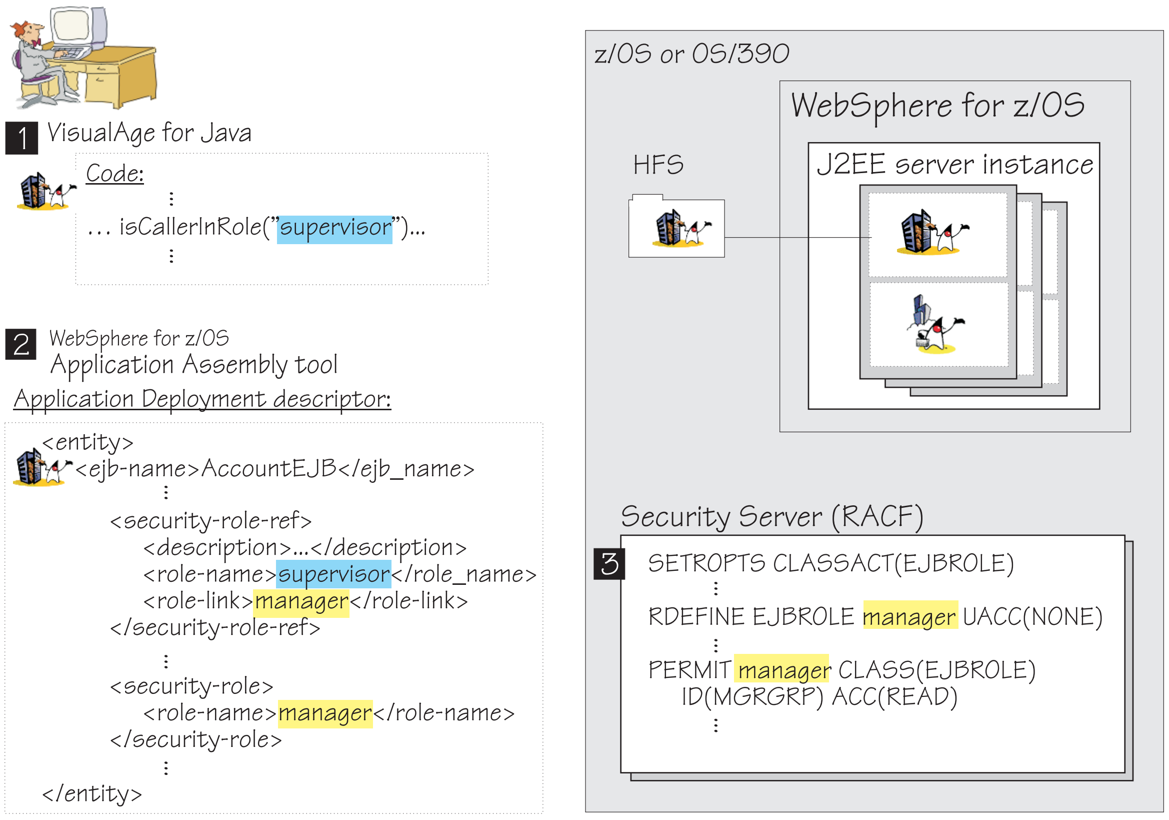

1 VisualAge for Java
Code:
... isCallerInRole("supervisor")...
2 WebSphere for z/OS Application Assembly tool
Application Deployment descriptor:
<entity>
<ejb-name>AccountEJB</ejb_name>
<security-role-ref>
<description>...</description>
<role-name>supervisor</role_name>
<role-link>manager</role-link>
</security-role-ref>
<security-role>
<role-name>manager</role-name>
</security-role>
</entity>
z/OS or OS/390
WebSphere for z/OS
HFS
J2EE server instance
Security Server (RACF)
3 SETROPTS CLASSACT(EJBROLE)
RDEFINE EJBROLE manager UACC(NONE)
PERMIT manager CLASS(EJBROLE) ID(MGRGRP) ACC(READ)

Figure 10. Deploying an Enterprise bean that uses security roles as authorization controls

The following list corresponds with the numbered items in Figure 10:

1. The sample Account bean uses the `isCallerInRole` interface to perform authorization checks for specific operations. For the `isCallerInRole` checks, the bean provider uses the role name `supervisor` for activities that might require authorization controls.

   Example: The code fragment illustrates a method for processing a deposit into the account. While several classes of user may invoke the deposit method, supervisor authority is necessary to deposit more than $10,000.

   ```
   public int deposit(int amount) throws java.rmi.RemoteException {
     if (amount > 10000) {
       if (!ejbContext.isCallerInRole("supervisor"))) {
       throw new SecurityException("Only a supervisor may process
           deposits of this amount");
       }
     }
     setBalance(Balance+amount);
     return Balance;
   }
   ```

   Bean providers must include security-role-reference elements when the `isCallerInRole` method is used in the component code. In the deployment descriptor, security role references may be defined on a component or method level.

2. The application assembler defines a security view for the banking application, with the help of the z/OS or OS/390 security administrator. For this sample, assume that the application assembler and security administrator decide that three RACF profiles are required for use with the `Account` bean: `customer`, `teller`, and `manager`.

Using the WebSphere for z/OS Application Assembly tool, the application assembler:

a. Adds one security role for each of the three z/OS or OS/390 (RACF) profiles, using role names that exactly match the profile names: `customer`, `teller`, and `manager`.

b. Links the bean's security role reference, `supervisor`, to the RACF profile, `manager`, which most closely matches the intent of the bean developer.

c. Defines method permissions for each of the bean methods. For example, you might define permissions as follows:
   - The `customer`, `teller`, and `manager` security roles may access the `deposit` method.
   - Only the `manager` role may access the `closeAccount` method.

Part of the resulting application deployment descriptor is shown in Figure 10 on page 28.

Note: The application assembler also has the option of explicitly defining the security identity under which individual methods or beans should run. Methods or beans may be run as client, server, or security role. For further information, see "RunAs identities" on page 30.

3. The z/OS or OS/390 security administrator needs to link the application assembler's security roles to actual users or user groups on z/OS or OS/390. For this sample, the security administrator:

   a. Activates the EJBROLE class and allows z/OS or OS/390 systems to share the generic profiles for this class.

   b. Defines the `customer`, `teller`, and `manager` profiles associated with the EJBROLE class, with universal access authority for each profile set to `NONE`.

   c. Permits access to each EJBROLE class profile to specific user IDs or groups, with `READ` access.

   d. Refreshes the EJBROLE class profile information stored in the RACF database.

Example:

```
SETROPTS CLASSACT(EJBROLE)
SETROPTS RACLIST(EJBROLE) GENERIC(EJBROLE)
/************************************************************/
/*  Defining EJBROLE profiles                               */
/************************************************************/
RDEFINE EJBROLE customer UACC(NONE)
RDEFINE EJBROLE teller UACC(NONE)
RDEFINE EJBROLE manager UACC(NONE)
/************************************************************/
/* Permitting  EJBROLE class access.                        */
/************************************************************/
PERMIT customer CLASS(EJBROLE) ID(PUBGRP) ACC(READ)
PERMIT teller CLASS(EJBROLE) ID(TELLRGRP) ACC(READ)
PERMIT manager CLASS(EJBROLE) ID(MGRGRP) ACC(READ)
SETROPTS RACLIST(EJBROLE) GENERIC(EJBROLE) REFRESH
```

When the sample application is installed in the WebSphere for z/OS J2EE server, and a client request drives the Account bean's `closeAccount` method, the J2EE server performs an authorization check on the caller's identity. If the client user ID is defined to the RACF group `MGRGRP`, this caller has RACF authority to the EJBROLE profile, and passes the authority check. WebSphere for z/OS will process the method request.

Alternative: To save some effort, especially in a more elaborate application than this one, the security administrator could group logical security roles in a RACF GEJBROLE profile. For example, the administrator could define a GEJBROLE profile `authorizedPersonnel` and allow certain methods, such as `deposit` to be accessed by the `authorizedPersonnel` security role. Then the administrator could add `teller` and `manager` as members of the `authorizedPersonnel` profile. Sample RACF commands for this alternative are as follows.

Example:

```
/*************************************************************/
/* Defining GEJBROLE profiles                                */
/*************************************************************/
SERTOPTS CLASSACT(GEJBROLE)
RDEFINE GEJBROLE authorizedPersonnel UACC(NONE) ADDMEM(teller manager)
```

For detailed instructions on deploying Enterprise beans that use security roles, see Chapter 11, "Using security roles and RunAs identities with Enterprise beans," on page 231.

RunAs identities

WebSphere for z/OS provides support for the use of "RunAs" identities, which is similar to the RunAs support defined in the Sun Microsystems EJB 2.0 specification. The WebSphere for z/OS RunAs support allows application assemblers to explicitly define the security identity under which individual methods should run. Methods may be run as caller, server, or security role. These identity settings control access to any required J2EE resources during execution of a J2EE application component or individual method. By default, during the execution of an Enterprise bean, WebSphere for z/OS uses the caller identity on downstream processing requests for J2EE resources. Downstream processing includes outbound calls to other J2EE servers (if the bean becomes a client of a bean residing in another server).

Using RunAs server or security role might simplify the set-up required to implement security for J2EE applications on the z/OS or OS/390 platform, but might not satisfy the needs of your J2EE application. For example, using the server region identity might be advantageous if your installation has a small number of WebSphere for z/OS J2EE servers handling requests from hundreds or thousands of different J2EE clients. In this case, your installation could grant authorization to only its server region user IDs, rather than granting authorization to all potential J2EE clients. On the other hand, your installation might want to track the use of system resources by specific clients or groups. In such a case, for accounting purposes, your installation might want to associate client user IDs or security roles with application processing.

To modify the default RunAs caller identity setting:

1. Use a WebSphere application assembly tool to modify the RunAs permission for a specific method, from caller to either server or security role. "A WebSphere application assembly tool" means either the z/OS edition of the Application Assembly tool, the WebSphere Studio Application Developer tool, the WebSphere Studio Application Developer Integration Edition tool, or Direct Deployment Tool/390fy.
2. Make sure that appropriate RACF profiles are in place for the server identity or security role identity to be authorized to perform downstream processing.

   For RunAs support, the security administrator defines profiles associated with the EJBROLE class, specifying application data, which is set to the actual user ID that WebSphere for z/OS is to use for the RunAs identity. For example, if you wanted to use RunAs identities for those bean methods that require management authority in the banking example described in “Security roles and method permissions” on page 27, you would change the RACF commands to include the highlighted APPLDATA parameter:

```
SETROPTS CLASSACT(EJBROLE)
SETROPTS RACLIST(EJBROLE) GENERIC(EJBROLE)
/*************************************************************/
/*  Defining EJBROLE profiles                                */
/*************************************************************/
RDEFINE EJBROLE customer UACC(NONE)
RDEFINE EJBROLE teller UACC(NONE)
RDEFINE EJBROLE manager UACC(NONE) APPLDATA('BANKMGR')
/*************************************************************/
/* Permitting  EJBROLE class access.                         */
/*************************************************************/
PERMIT customer CLASS(EJBROLE) ID(PUBGRP) ACC(READ)
PERMIT teller CLASS(EJBROLE) ID(TELLRGRP) ACC(READ)
PERMIT manager CLASS(EJBROLE) ID(MGRGRP) ACC(READ)
SETROPTS RACLIST(EJBROLE) GENERIC(EJBROLE) REFRESH
```

3. Use the WebSphere for z/OS Administration application to install the application, and activate the J2EE server.

Chapter 11, “Using security roles and RunAs identities with Enterprise beans,” on page 231 contains detailed instructions for deploying Enterprise beans that use RunAs identities.

Synchronizing operating system thread identity to RunAs identity: WebSphere for z/OS additionally allows application assemblers and deployers to associate the RunAs identity with the operating system thread, by setting the `Set OS thread identity to RunAs identity` for specific bean methods. This association means that the caller or security role identity— rather than the server region identity— is used for z/OS or OS/390 system service requests, such as access to files and database management systems. Note that the WebSphere for z/OS J2EE server may be configured to enable or disable this association (or synchronization). The default setting disables the ability to modify the identity on the operating system thread, regardless of the `Set OS thread identity to RunAs identity` setting in the installed application’s deployment descriptor. If the application installer does not enable synchronization, any method that sets the RunAs identity to the operating system thread will fail with a `no_permission` error.

To force WebSphere for z/OS to use the RunAs identity on the operating system thread:

1. Use the WebSphere for z/OS Application Assembly tool to set the `Set OS thread identity to RunAs identity` property for a specific method. (Clicking on the IBM `+ThreadID` tab opens a display window that lists each method in the bean, with checkboxes to the left of each method name. Clicking on the checkbox selects the method for synchronizing to the operating system thread.)
2. Make sure that appropriate RACF profiles are in place for the server identity or security role identity to be authorized to access z/OS or OS/390 system resources.
3. Use the WebSphere for z/OS Administration application to:
   a. Configure the J2EE server to enable setting the operating system thread to the RunAs identity.
   b. Install the application.
   c. Activate the new or modified J2EE server configuration.

Summary of RunAs and operating system thread settings and behavior: Table 3 illustrates how various combinations of RunAs and operating system thread settings in an application and J2EE server affect run-time processing.

Table 3. Summary of RunAs and OS thread settings and behavior

Deployment descriptor contents		**Setting for J2EE server property ″Enable Setting OS Thread ID to RunAs ID″**	
RunAs setting	**Set OS thread identity to RunAs identity setting**	**Disabled (default)**	**Enabled**
RunAs server	Not selected	**Authorization checks:** Use J2EE server region identity **Method execution and downstream calls:** Use J2EE server region identity **System services:** Use J2EE server region identity	Same behavior as listed for disabled (default) setting for J2EE server property.
	Selected	`No_permission` failure when method is invoked	Same behavior as listed for disabled (default) setting for J2EE server property.
RunAs caller (or no explicit RunAs setting)	Not selected	**Authorization checks:** Use caller's user ID (usually client user ID) **Method execution and downstream calls:** Use J2EE server region identity **System services:** Use J2EE server region identity	Same behavior as listed for disabled (default) setting.
	Selected	`No_permission` failure when method is invoked	All processing uses caller's user ID.
RunAs *security role*	Not selected	**Authorization checks:** Use RACF ID that is mapped to specified security role **Method execution and downstream calls:** Use J2EE server region identity **System services:** Use J2EE server region identity	Same behavior as listed for disabled (default) setting.
	Selected	**Authorization checks:** Use RACF ID that is mapped to specified security role **Method execution and downstream calls:** Use RACF ID that is mapped to specified security role **System services:** Use J2EE server region identity	All processing, including system services, uses RACF ID that is mapped to specified security role.

Given the information presented in Table 3 on page 32, you can determine what security mechanisms you need to have in place for Enterprise beans that use security roles or identities. See “Summary of requirements for using security roles and identities” for further details.

Summary of requirements for using security roles and identities

Table 4 briefly reviews the assembly and run-time configuration requirements for deploying Enterprise beans that use programmatic security controls. For detailed instructions for deploying such beans, see Chapter 11, “Using security roles and RunAs identities with Enterprise beans,” on page 231.

Table 4. Assembly and configuration tasks for programmatic security controls

If your application uses. . .	**Then complete these application assembly tasks. . .**	**And complete these run-time configuration tasks. . .**
isCallerInRole method	1. Define security roles 2. Link security role references to the defined roles 3. Modify method permissions by selecting appropriate roles	1. Define EJBROLE classes and profiles using RACF 2. (Optional) Define GEJBROLE classes **Note:** During run-time, the isCallerInRole method always returns `true` if the EJBROLE class has not been activated.
Security roles and method permissions	1. Define security roles 2. Modify method permissions by selecting appropriate roles	1. Define EJBROLE classes and profiles using RACF 2. (Optional) Define GEJBROLE classes **Note:** During run-time, method permission checks always return `true` if the EJBROLE class has not been activated.

Table 4. Assembly and configuration tasks for programmatic security controls (continued)

If your application uses. . .		Then complete these application assembly tasks. . .	And complete these run-time configuration tasks. . .
RunAs settings	server	No assembly tasks are required, beyond setting the RunAs setting.	Make sure that the J2EE server region user ID has authorization to use system resources that the bean requires. For example, check: • CB.*srvname* profile for the CBIND class • Authentication credentials for communicating with remote servers.
	caller (or no explicit setting)	No assembly tasks are required	Make sure that the J2EE client's user ID has authorization to use system resources that the bean requires. For example, check: • DB2 GRANT statements or DSNR class profile • Authentication credentials for communicating with remote servers.
	security role	1. Define security roles 2. Modify RunAs settings by selecting appropriate roles (assign user ID for method execution)	Make sure that the RACF ID that is mapped to the security role has authorization to use system resources that the bean requires. For example, check: • DB2 GRANT statements or DSNR class profile • Authentication credentials for communicating with remote servers.
Set OS thread identity to RunAs identity		No assembly tasks are required, beyond setting the Set OS thread identity to RunAs identity property.	1. Make sure that the identity used on the thread has authorization to use any system resources. 2. Configure the J2EE server to enable synchronizing on the operating system thread.

Overview of SQLID for managed datasources

SQLID for managed datasources is the WebSphere for z/OS equivalent of userid/password for controlling the use of unqualified table references in container-managed (CMP) and bean-managed (BMP) Enterprise beans.

WebSphere for z/OS applications that access relational data are frequently implemented with SQL statements containing unqualified table references. These unqualified table references must be resolved at runtime. CMP entity beans and J2EE components that perform direct JDBC access may be constructed with unqualified table references. The use of unqualified table references promotes application portability by not constraining the implementation to the name space conventions of a particular database environment.

WebSphere Application Server enables an application deployer to control the table qualifier used by the database at runtime. This resolves unqualified table references by defining a qualifier as part of a managed datasource definition through the Administration application. This qualifier is specified as a userid/password pair as part of a datasource definition on all WebSphere platforms except z/OS and OS/390. On z/OS and OS/390, the qualifier is specified as an SQLID.

SQLID for managed datasources is a technique used to control effective qualifier references in DB2 unqualified database table references. These references take the form of `<qualifier>.<tablename>` where the `<qualifier>` exists so that tables with the same name can exist in the same database.

Example: Here are two examples of database table references:
- `test.customer`
- `prod.customer`

The full SQLID statement would therefore take the following form:

```
SELECT * FROM [qualifier].ATABLE
```

The technique, which is based on the `SQL SET CURRENT SQLID` statement, was developed as an alternative to the WebSphere Application Server Advanced Edition userid/password function.

Note: See *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834 for more information.

Qualified and unqualified table references

SQL statements are written with either qualified or unqualified table references. If the table reference is qualified, it is absolute. If unqualified, the database applies a set of rules (see below) to determine the qualifier.

Rules for CMP: These rules determine the SQL statement qualifier in an unqualified database table reference for CMP. Going down the list, the first one that you have will determine your SQL statement qualifier:
- SQLID specified on a datasource definition.
- The userid associated with the current environment (e.g. address space identity).

Rules for BMP: These rules determine the SQL statement qualifier in an unqualified database table reference for BMP. Going down the list, the first one that you have will determine your SQL statement qualifier:
- The userid established at connection when the connection is obtained with a userid and password.
- SQLID specified on a datasource definition.
- The userid associated with the current environment (e.g. address space identity).

Note: See *DB2 Application Programming and SQL Guide*, SC26-9933 for more information.

Application Developer Integration Edition schema mapping editor

The WebSphere Studio Application Developer Integration Edition schema mapping editor makes it possible to assign either unqualified table references in the SQL statements (which are generated to support the persistence of the CMP entity bean) or a single, statically-bound qualifier.

Note: If you are doing direct JDBC programming in a component, you get to decide whether you want to use qualified or unqualified table references.

Web Security

As a component of the Secure Way Security Server for z/OS and OS/390, RACF provides the functions of authentication and access control for z/OS and OS/390 resources and data (see *z/OS Security Server RACF Security Administrator's Guide*, SA22-7683). In addition, WebSphere for z/OS provides Web security for applications running under its control. This application security is administered using a WebSphere application assembly tool. "A WebSphere application assembly tool" means either the z/OS edition of the Application Assembly tool, the WebSphere Studio Application Developer tool, the WebSphere Studio Application Developer Integration Edition tool, or Direct Deployment Tool/390fy.

Similar to EJB access, access to Web applications is controlled by associating the applications with roles, for which security has been defined. As with EJB security, Application Assemblers use the Application Assembly tool for z/OS to specify the security policy to be applied to a Web application. Specifically, when installing a Web application into a Web container, they specify what roles are allowed to access a particular URL by specifying security-constraints in the Web application's deployment descriptors.

The following example shows a web.xml file, containing security constraints, that was generated when roles were specified:

```
<!DOCTYPE web-app (View Source for full doctype...)>
<web-app>
   <display-name>SecureBasic</display-name>
   <description>SecureBasic Webapp</description>
   <servlet>
      <servlet-name>HelloWorldSecureBasicServlet</servlet-name>
      <display-name>Hello World Secure Basic Servlet</display-name>
      <description>Hello World Secure Basic Servlet</description>
      <servlet-class>HelloWorldSecureBasicServlet</servlet-class>
      <security-role-ref>
          <role-name>manager</role-name>
          <role-link>manager</role-link>
      </security-role-ref>
   </servlet>
   <servlet>
      <servlet-name>HelloSecureBasicSessionServlet</servlet-name>
      <display-name>HelloSecureBasicSessionServlet</display-name>
      <description>HelloSecureBasicSessionServlet</description>
      <servlet-class>HelloSecureBasicSessionServlet</servlet-class>
      <security-role-ref>
           <role-name>manager</role-name>
           <role-link>manager</role-link>
      </security-role-ref>
  </servlet>
  <servlet-mapping>
     <servlet-name>HelloSecureBasicSessionServlet</servlet-name>
     <url-pattern>/helloSBsession</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
     <servlet-name>HelloWorldSecureBasicServlet</servlet-name>
     <url-pattern>/helloSBworld</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
     <welcome-file>index.html</welcome-file>
  </welcome-file-list>
  <security-constraint>
     <web-resource-collection>
         <web-resource-name>SecureBasicResourceCollection</web-resource-name>
         <description>SecureBasic Webapp Resource Collection</description>
      <url-pattern>/helloSBsession</url-pattern>
      <url-pattern>/helloSBworld</url-pattern>
```

```
         <http-method>GET</http-method>
         <http-method>PUT</http-method>
      </web-resource-collection>
      <auth-constraint>
         <description />
         <role-name>manager</role-name>
      </auth-constraint>
   </security-constraint>
      <login-config>
         <auth-method>BASIC</auth-method>
         <realm-name>SecureBasicWebappRealm</realm-name>
      </login-config>
      <security-role>
         <description />
         <role-name>manager</role-name>
      </security-role>
      <security-role>
         <description />
         <role-name>employee</role-name>
      </security-role>
      <security-role>
         <description>Big Boss</description>
         <role-name>bigboss</role-name>
      </security-role>
   </web-app>
```

For complete information on specifying security constraints in a Web application, see the JAVA Servlet Specification v2.2, available at:

`http://java.sun.com`

When processing a request, the Web container understands what roles, if any, are required to access the component represented by the input URL. The container will then validate that the requestor has been authenticated and that the authenticated user has been granted permission to the required roles. The Web container makes use of the same SAF based User registry and EJB role profiles as the EJB cntainer to perform this validation. Therefore, you can use the same User Registry and role profiles for administering Web applications as you use for Enterprise Beans and J2EE Services.

Authenticating Web Clients

The main difference between setting up EJB security and setting up Web security is the challenge mechanism used to acquire authentication data from a requestor. The authentication data for requestor of a Web application can be specified within the deployment descriptors of that Web application. The Application Assembler and System Administrator can use a WebSphere application assembly tool to provide these deployment descriptors during the assembly process. “A WebSphere application assembly tool” means either the z/OS edition of the Application Assembly tool, the WebSphere Studio Application Developer tool, the WebSphere Studio Application Developer Integration Edition tool, or Direct Deployment Tool/390fy.

The following challenge mechanisms are suppported:

HTTP Basic Authentication
: If this is specified, WebSphere for z/OS will challenge the requestor for a userid and password using HTTP Basic Authentication. This is similar to the function provided in Web servers, such as Apache and IBM HTTP Server. HTTP Basic Authentication does not address the issue of securing the transport of the userid and password between the server and client. As with all other servers, it is the responsibility of the application provider

and the administrator to structure the application so that basic authentication is only performed over a secure connection such as HTTPs and SSL.

Client Certificate
If this is specified, access to a resource requires a user provided client certificate. WebSphere for z/OS will ensure that the request is being made over a mutually authenticated SSL connection. If the mutual authentication is successfully completed between the server and the HTTP client, WebSphere for z/OS will consider the certificate to be authentic.

Custom Form
If this is specified, a custom login page must be provided with the application. When WebSphere for z/OS determines that it needs to authenticate a user, it will present the custom form to the user's browser to prompt for a userid and password.

This challenge mechanism adheres to the JAVA Servlet Specification V2.2 requirement that Web containers support the capability to manage a Form-based Login. This specification describes how to provide the form and how to include it in the deployment descriptor of the Web application.

When a user is authenticated to an application via a Form-Based Login, the Web container creates a secure Login Token that is valid for a configurable period of time (see Chapter 14, "Steps for configuring Web security," on page 247). The login is valid on any requests to this Web application on any J2EE server within the same z/OS or OS/390 sysplex. The Login Token is communicated to the browser via HTTP Cookies. The cookie containing the Login Token is sent by the browser on subsequent requests to this Web application. The Web container uses the Login Token on each request to determine if the login is still valid or if the user needs to be re-authenticated.

Login Tokens do not contain sensitive authentication data such as passwords. The Tokens are encrypted using private keys that are maintained in a Server Key ring that can be configured by the administrator to only be accessible by specific WebSphere Application Server instances. In addition, the administrator has the ability to specify that Login Tokens are only to be communicated via a secure transport. When this option is requested, the Web container will ensure that the response is being sent over a secure communication channel such as SSL, before including the cookie containing the Login Token in the output stream. In addition, the Web container will set the secure bit in the cookie which indicates to the browser to only send the cookie on subsequent requests that are made over an SSL connection. Cookies containing Login Tokens are created in a manner that instructs the browsers only to maintain them in its session cache and to not save them on disk.

For example, in a normal situtation, when the LoginTokenEncrypt environment variable is set to true, the following series of events would occur:

1. The client issues an HTTP request for which Form-based authentication is required.
2. The Web container saves the original request in an original request cookie (jwwwrequest cookie), gets the Form-based Login page, and serves the Form-based Login page to the requesting client.
3. The Form-based Login page is displayed on the client's browser.
4. The client enters a user ID and password.

5. If the user ID and password are valid, the Web container creates a Login Token cookie ((jwwwcontent cookie), in which the client's user ID is encrypted. It then retrieves the original request and sends the original request and the Login Token cookie to the J2EE server for processing.
6. The J2EE server decrypts the client's user ID, does a surrogate login for the user with the identity of the address space, and serves the original requested page.
7. The Login Token cookie is attached to any subsequent request from the same client, thereby eliminating the need for the client to re-enter his user ID and password.

Note: If you are using an IBM HTTP Server, along with the Local Redirector Plug-in to handle HTTP requests, (i.e., the WEB_SECURITY_VERSION property in the jvm.properties file is set to 1 or the property is absent), some of the actions noted above as being performed by the Web container, are performed by the Local Redirector Plug-in.

Digest Authentication
WebSphere does not support this method of authentication on any platform at this time.

Single sign-On

A Login Token can be used for multiple applications existing on different WebSphere Application Server's serving as virtual hosts, provided these virtual hosts reside in the same sysplex within the domain specified in the webcontainer.conf file (see Chapter 14, "Steps for configuring Web security," on page 247). The name of this domain is used when HTTP cookies are created for the single sign-on, and determines the scope to which a single sign-on applies.

Note: Cross-domain Single Sign-On is not supported.

Selecting a Web container security collaborator level

The security functions the Web container can provide is determined by the version of the Web container security collaborator that is specified in the `jvm.properties` file:

- Version 1 of the Web container security collaborator uses a SAF user registry and only provides the following security functions for requests received by the IBM HTTP Server for z/OS and forwarded to the Web container via the WebSphere for z/OS Local Redirector plug-in. None of these functions were available for requests received by the HTTP or HTTPS Transport Handlers:
  - Basic authentication
  - Form-Based authentication
  - Client Certificates
  - Single Sign-On across WebSphere/390 Servers
- Version 2 of the Web container security collaborator enables the Web container to provide most of these security functions for requests that are received by the HTTP or HTTPS Transport Handler as well as for requests received by the IBM HTTP Server for z/OS. This version of the collaborator also enables you to use a trust association interceptor with WebSphere for z/OS.

The following table summarizes the capability and configuration requirements for the version 1 and Version 2 web security collaborators.

Table 5. Summary of the two Versions of the Web container security collaborator

	Version 1	**Version 2**
Security functions supported	• Basic authentication • Form-Based authentication[1] • Client certificate authentication • Single sign-on authentication across IBM HTTP Servers for z/OS[1]	• Basic authentication • Form-Based authentication[2] • Single sign-on across IBM HTTP Servers for z/OS[2] • Trust asssociation interceptor[3]
Security is applied to requests received via	IBM HTTP Server for z/OS and forwarded to the Web container via the WebSphere for z/OS Local Redirector plug-in.	• HTTPTransport Handler • HTTPS Transport Handler • IBM HTTP Server for z/OS and forwarded to the Web container via the WebSphere for z/OS Local Redirector plug-in.
Enabled by	Specifying WEB_SECURITY_VERSION=1 in the JVM properties file or by not including a WEB_SECURITY_VERSION property in the JVM properties file (1 is the default value).	Specifying WEB_SECURITY_VERSION=2 in the JVM properties file.

Table 5. Summary of the two Versions of the Web container security collaborator (continued)

Notes:

1. To enable Form-Based authentication or single sign-on capability for Web applications being received by the IBM HTTP Server for z/OS, working in conjunction with the WebSphere for z/OS Local Redirector Plugin, you must:
   - Set the **WebAuth.LoginToken.Encrypt** property in the webcontainer.conf file to true.
   - Create a profile of the form BPX.SRV.<userid> in the SURROGAT class for each userid that should be able to login using Form-Based authentication. (The profile BPX.SRV.* may be used if all users should be able to login.)
   - Grant the IBM HTTP Server's userid read access to this profile.
   - Set the **JAVA_PROPAGATE** variable In the IBM HTTP Server for z/OS's httpd.envvars file to NO.

   Additionally, if you are going to be using ICSF, you must:
   - Create ICSF keys and grant the IBM HTTP Server for z/OS's address space access to them.
   - Set the **WebAuth.EncryptionKeyLabel** property in the webcontainer.conf file to label of the cryptographic key that is to be used for Web application security.

   If you are not going to be using ICSF, you must:
   - Set the **WEB_SECURITY_VERSION** property in the jvm.properties file to 1.
   - Set the following properties in the webcontainer.conf file:
     - The **WebAuth.UnauthenticatedUserSurrogate** property must be set to the identity under which the HTTP Server runs.
     - The **WebAuth.SingleSignOn.Enabled** property must be set to false.
     - The **WebAuth.LoginToken.Encrypt** property must be set to false.
2. To enable Form-Based authentication or single sign-on capability for Web applications being received by the HTTP or HTTPS Transport Handler, you must:
   - Set the following properties in the webcontainer.conf file:
     - The **WebAuth.EncryptionKeyLabel** property must specify the label of the cryptographic key that is to be used for Web application security.
     - The **WebAuth.LoginToken.Encrypt** property must be set to true.
   - Add the **WEB_SECURITY_VERSION** property to the jvm.properties file and set it to 2.
   - Create ICSF keys and make them available to the server region.
   - Permit the server region user ID to the CSFSERV general resource class.
   - Add the **ENABLE_TRUSTED_APPLICATIONS** environment variable to your J2EE server's current.env. file, and set it to 1.
3. To enable trust association interceptor support, you must:
   - Make the following changes to your J2EE server's current.env file:
     - Add the **ENABLE_TRUSTED_APPLICATIONS** environment variable to your J2EE server's current.env. file, and set it to 1.
     - Add the **ENABLE_TRUSTED_APPLICATIONS**TrustAssociationInterceptor class to the CLASSPATH environment variable.
   - Add the following properties to the WebSphere for z/OS webcontainer.conf configuration file:
     - **WebAuth.TrustAssociationInterceptor.**<value>**.ImplClass=**<classname>
     - **WebAuth.TrustAssociationInterceptor.**<value>**.Properties=**<filename>
   - Add the **WEB_SECURITY_VERSION** property to the jvm.properties file and set it to 2.

Naming

WebSphere for z/OS uses the LDAP component of the z/OS or OS/390 Security Server to provide naming and directory services. When your system programmers install and customize WebSphere for z/OS, they set up an LDAP server that uses DB2 tables to store name and directory information.

Within the WebSphere for z/OS naming system, there are two namespaces:

- The global namespace, which initially contains home and object references for J2EE application components that are installed in J2EE servers.

  Your installation may use LDAP access control lists (ACLs) to control access to the global namespace by J2EE servers, application components and clients. The global namespace is accessible by a program in any Java Virtual Machine (JVM), z/OS or OS/390 system, or J2EE server region as long as the program has appropriate authorization.
- The local namespace, which contains naming subcontexts represented through `java:comp`. The local namespace is accessible only by the Java programs running within a single JVM. In other words:
  - All J2EE application components that run in the same J2EE server region may access the same local namespace.
  - A single Java client running on z/OS or OS/390 may access its local namespace.

WebSphere for z/OS uses caching to increase the performance of JNDI lookup operations related to both the global and local namespaces. To speed up subsequent lookups, WebSphere for z/OS caches JNDI context object as they are bound or initially looked up.

A Java client running on z/OS or OS/390 may change the caching behavior related to only the global namespace, using one of the following:

- An environment Hashtable in the client code.
- A `jndi.properties` resource file.

  Note: Because a default JNDI properties file is shipped with WebSphere for z/OS, Java clients must ensure that their own JNDI properties file appears in the CLASSPATH before the default file. In other words, the client's properties file must precede any WebSphere for z/OS JAR files that the client uses (for example: `ws390crt.jar`).
- The java command line, using the -D switch.

Before changing JNDI caching, however, you should understand the default caching behavior in more detail. The following information describes only the JNDI caching behavior related to the global namespace. WebSphere for z/OS naming services associates a cache with the initial context (when a `javax.naming.InitialContext` object is instantiated), and searches the environment properties for a cache name. The default cache name is either the provider URL, or `iiop:///` if no provider URL is defined.

Also by default, each cache and its entries have a maximum life that is limited to the life of a JVM process (that is, the life of a J2EE server region or Java client). You may, however, use a caching property to clear cache entries before the maximum life of either the cache or the cache entries is reached.

Notes:

1. WebSphere for z/OS evaluates cache properties whenever an InitialContext instance is created.
2. All instances of InitialContext that use the same cache name within a particular JVM share the same cache instance.
3. After an association between an InitialContext instance and cache is established, the association does not change. A javax.naming.Context object returned from a lookup operation inherits the cache association of the Context object on which the lookup was performed.
4. Changing cache property values with the `Context.addToEnvironment` or `Context.removeFromEnvironment` method does not affect cache behavior. Properties affecting a given cache instance, however, may be changed with each InitialContext instantiation.

Further information related to WebSphere for z/OS naming services appears in the following places:

- "Java Naming and Directory Interface™ (JNDI)" describes JNDI programming considerations and JNDI properties related to caching behavior.
- Chapter 10, "Using JNDI look-ups," on page 227 provides programming examples using JNDI calls.
- *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834 contains recommendations and instructions for configuring the global namespace, and for configuring an LDAP server and DB2 for use with WebSphere for z/OS J2EE servers.

Application programming interfaces

The following topics briefly describe the technologies and associated application programming interfaces, in the context of using them in the WebSphere for z/OS environment. If you are unfamiliar with the concepts and terms in these topics, consider reading the tutorials, specifications, and other general documentation which is available at the Sun Microsystems Web site (`http://java.sun.com`).

Java Naming and Directory Interface™ (JNDI)

To access WebSphere for z/OS naming services, J2EE application components and clients set the `java.naming.factory.initial` property to the `com.ibm.websphere.naming.WsnInitialContextFactory` value, and then use JNDI calls to use naming services.

A Java client running on z/OS or OS/390 may change the caching behavior related to only the global namespace, using one of the following:

- An environment Hashtable in the client code.
- A `jndi.properties` resource file.
- The java command line, using the -D switch.

To change cache behavior in a Java client running on z/OS or OS/390, use the following JNDI caching properties. Keep in mind that these properties modify caching behavior for the individual J2EE application, not for all applications running in a WebSphere for z/OS J2EE server.

com.ibm.websphere.naming.jndicache.cachename=providerURL
Specifies the name of the cache that a J2EE server uses for Java Naming and Directory Interface (JNDI) lookups for the global namespace. Use this property

only if you want to assign a cache name other than the default value of the provider URL. Valid options for cache names are:

providerURL
Specifies the default cache name, which is the same as the value for the `java.naming.provider.url` property. URLs are normalized by stripping off everything after the port. For example, `iiop://server1:900` and `iiop://server1:900/com/ibm/initCtx` are normalized to the same cache name.

If no value is supplied for the `java.naming.provider.url` property, the J2EE server uses the value `iiop:///` as the default cache name.

any string
Specifies a string as the cache name. Any arbitrary string may be used as a cache name.

Note: Whenever an InitialContext instance is created, WebSphere for z/OS evaluates all JVM properties related to JNDI caching.

com.ibm.websphere.naming.jndicache.cacheobject=populated
Specifies whether a J2EE server uses caching for Java Naming and Directory Interface (JNDI) lookups for the global namespace, or specifies when to clear an existing cache. Valid values for this property are:

populated
Specifies that JNDI caching should occur and, if a JNDI cache with the same name already exists, any existing cache entries should be left in the cache. If a JNDI cache with the same name does not yet exist, the J2EE server creates a new cache.

cleared
Specifies that JNDI caching should occur and, if a JNDI cache with the same name already exists, any existing cache entries should be removed from the cache. If a JNDI cache with the same name does not yet exist, the J2EE server creates a new cache.

none
Turns off JNDI caching. If this option is specified, the cache name is irrelevant. Therefore, this option will not disable a cache that is already associated with other InitialContext instances. The InitialContext being instantiated will not be associated with any cache.

Note: Whenever an InitialContext instance is created, WebSphere for z/OS evaluates all JVM properties related to JNDI caching.

com.ibm.websphere.naming.jndicache.maxcachelife=0
Sets the maximum lifetime of a cache for Java Naming and Directory Interface (JNDI) lookups for the global namespace. Valid values are:

0 Sets the maximum lifetime of the JNDI cache to the life of the JVM process (that is, the life of a J2EE server region or Java client). Cached objects remain in the cache either until the process ends, or until WebSphere for z/OS evaluates the property setting `com.ibm.websphere.naming.jndicache.cacheobject=cleared` for this cache.

Positive integer
Sets the maximum lifetime of the cache, in minutes, to the specified value. When the maximum cache lifetime is reached, WebSphere for z/OS clears the cache before performing another cache operation. The cache is repopulated as bind, rebind, and lookup operations are executed.

Note: Whenever an InitialContext instance is created, WebSphere for z/OS evaluates all JVM properties related to JNDI caching.

com.ibm.websphere.naming.jndicache.maxentrylife=0
Set the maximum lifetime of individual cache entries in a cache for Java Naming and Directory Interface (JNDI) lookups for the global namespace. Valid values are:

0 Sets the maximum lifetime of JNDI cache entries to the life of the JVM process (that is, the life of a J2EE server region or Java client). Cached objects remain in the cache either until the process ends, or until WebSphere for z/OS evaluates the property setting `com.ibm.websphere.naming.jndicache.cacheobject=cleared` for this cache.

Positive integer
Sets the maximum lifetime of individual cache entries, in minutes, to the specified value. When the maximum lifetime for an entry is reached, the next attempt to read the entry from the cache will cause the entry to be refreshed.

Note: Whenever an InitialContext instance is created, WebSphere for z/OS evaluates all JVM properties related to JNDI caching.

Further information related to WebSphere for z/OS naming services appears in the following places:
- "Naming" on page 42 describes default JNDI caching behavior.
- Chapter 10, "Using JNDI look-ups," on page 227 provides instructions and programming examples related to using JNDI.

Java™ Message Service

The Java Message Service (JMS) provides a framework for developing and supporting Java software components that communicate by creating, sending, and receiving messages. This method of communication, known as messaging, allows components to interact asynchronously and reliably, without knowing more about their communication partners than message formats and destinations.

The Sun Microsystems J2EE specification defines basic JMS API concepts, which include the following:
- Two messaging domains: Point-to-point and publish/subscribe.
  - In the point-to-point domain, one message producer creates and sends messages to a queue, from which one message consumer retrieves the messages.
  - In the publish/subscribe domain, one message producer creates and sends messages to a topic, from which many message consumers retrieve the messages.

  WebSphere for z/OS supports both messaging domains.
- The JMS API architecture, which is composed of the following:
  - JMS clients, which are Java-language programs or components that produce and consume messages. In the WebSphere for z/OS environment, JMS clients may be any application components that run in the J2EE server.
  - A JMS provider, which implements the JMS interfaces and provides administrative and control features. WebSphere for z/OS uses IBM's MQSeries as a JMS provider.
  - Messages and administered objects, which are connection factories and message destinations. With WebSphere for z/OS, connection factories and

destinations are part of the configuration for a J2EE server. They represent MQSeries resources; that is, they enable JMS clients to access and use MQSeries services, queues and topics.

- The JMS API programming model, which describes the following components of a JMS application and how they work together:
  - Administered objects
  - Connections
  - Sessions
  - Message producers and consumers

  Application components running in a WebSphere for z/OS J2EE server follow the JMS API programming model, using the classes, interfaces and methods documented in *MQSeries Using Java*, SC34-5456.

Figure 11 illustrates sample WebSphere for z/OS configurations for JMS:

1. A point-to-point messaging domain, using MQSeries on z/OS or OS/390.
2. A publish/subscribe domain, which requires a message broker. You may use an off-platform message broker.

   Limitation: This domain requires MQSeries function that is not currently available in the MQSeries for OS/390 product. To participate in the publish/subscribe domain, the J2EE server's connection factory and destination resources must point to a local MQSeries on z/OS or OS/390, which, in turn, must be able to communicate with a queue manager that is hosting the message broker (in other words, an MQSeries on a non-z/OS or non-OS/390 platform).

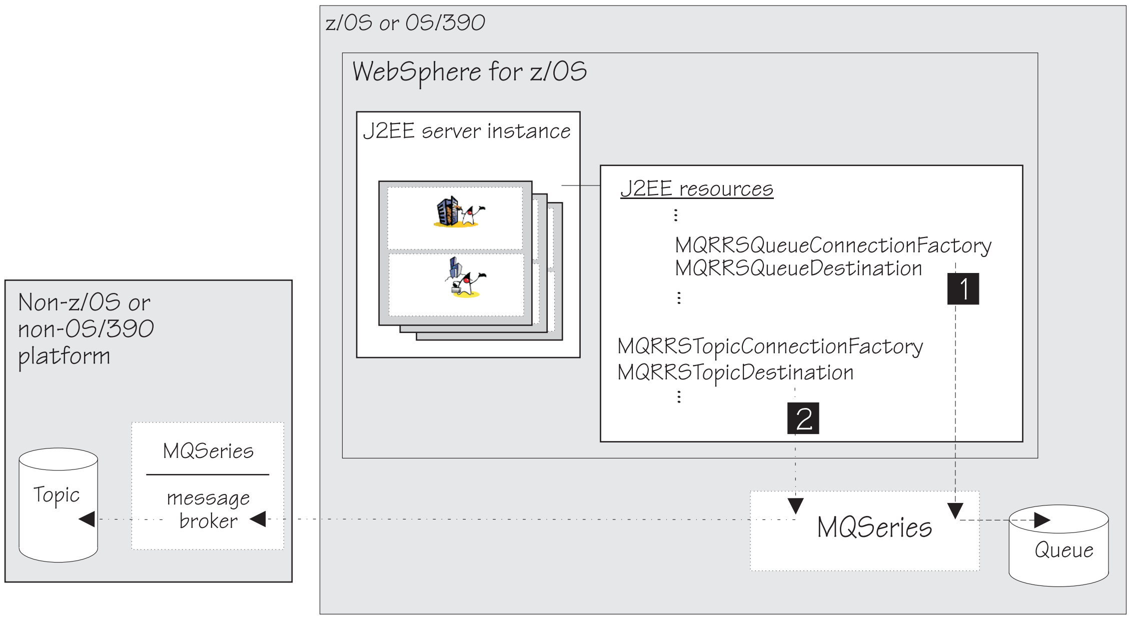


Figure 11. Sample JMS configurations for WebSphere for z/OS

Regardless of the type of domain in use, JMS messaging interactions follow this pattern:

1. A JMS client uses a JNDI lookup to find a connection factory. Through the returned reference, the client establishes access to the JMS provider.

2. The JMS client then uses a JNDI lookup for a destination; this action defines the target to which the client will direct its messages (or from which the client will retrieve messages). The client may define one or more target destinations.
3. Using the connection to the JMS provider, the JMS client creates one or more sessions to logically group its messaging activity.
4. With the connection, destination, and session references established, the JMS client creates and uses sessions to send or retrieve messages.

Three factors determine how WebSphere for z/OS manages the JMS client's sessions:

- The type of J2EE resources for JMS used in the WebSphere for z/OS J2EE server configuration,
- The transactional environment when the JMS client uses a session, and
- The `transacted` and `acknowledge` arguments specified on the `createQueueSession` or `createTopicSession` method in the JMS client code.

Selecting the type of ConnectionFactory to use for JMS

WebSphere for z/OS supports two types of ConnectionFactory for JMS: RRS-enabled and base. With the RRS-enabled type, WebSphere for z/OS treats sessions as protected resources, and participates in two-phase commit operations that can prevent the loss or corruption of data when system or application failures occur. WebSphere for z/OS can participate in such operations because of its resource recovery services (RRS) component. Generally speaking, using the RRS-enabled ConnectionFactory will result in behavior that most closely matches the intent of the application programmer who coded the JMS client, so most WebSphere for z/OS J2EE server configurations should use this RRS-enabled type.

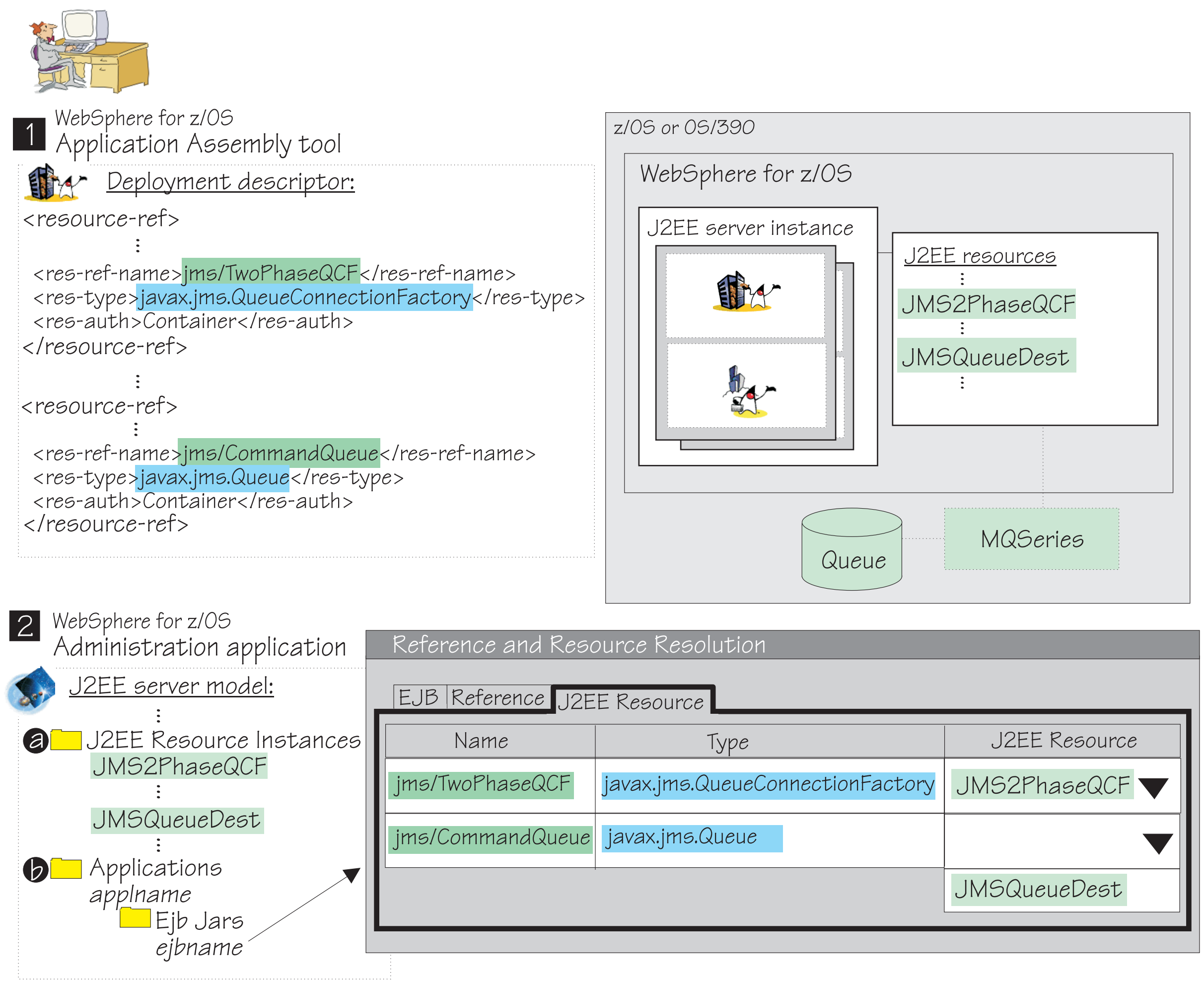
Table 6 on page 48 summarizes the two types of ConnectionFactory and how they handle sessions, based on the JMS client's transactional context and the sttributes specified on the `create...Session` method in the client code.

Table 6. Types of J2EE resources for JMS and their behaviors

Type of Connection Factory	**Transactional environment when JMS client uses session**	**Attributes on `create...Session` method**		**Behavior**
RRS-enabled	A global transaction is in progress	Both transacted attribute and acknowledge mode are ignored		Messages issued or retrieved during a session are not sent or delivered until the global transaction is committed (if the global transaction is rolled back, messages are not sent at all)
	No global transaction is in progress	transacted attribute is `true`	acknowledge mode is ignored	Behavior is the same as listed below for the base type, for each combination of attributes, with the following exception: WebSphere for z/OS will check transaction boundaries. For example, an Enterprise bean that manages its own transactions (a BMT bean) might try to do the following: 1. Send several messages under a local transaction 2. Start a global transaction for a sequence of operations In such a case, WebSphere for z/OS will not allow the BMT bean to start the global transaction because of the incomplete messaging interactions issued under the local transaction. In other words, the BMT bean cannot start a global transaction because it did not explicitly commit the local transaction that was pending on the session.
		transacted attribute is `false`	acknowledge mode determines behavior	
Base	Ignored; only the session's attributes affect behavior	transacted attribute is `true`	acknowledge mode is ignored	Messages issued or retrieved during a session are not sent or delivered until the session is committed. If the session is rolled back, messages are not sent at all.
		transacted attribute is `false`	acknowledge mode determines behavior	Messages are issued or retrieved immediately for the `AUTO_ACKNOWLEDGE` and `DUPS_OK_ACKNOWLEDGE` modes. For `CLIENT_ACKNOWLEDGE` mode, messages are delivered immediately after their `acknowledge` method is driven.

Linking JMS resource references to a WebSphere for z/OS configuration

To enable the use of the JMS API for application components running in WebSphere for z/OS, you need to link the WebSphere for z/OS connection factories and destinations to the equivalent JMS resource references that your application's components use. Figure 12 on page 49 provides an overview of the steps required to complete this link; step-by-step instructions appear in Chapter 12, "Using the Java Message Service API in J2EE application components," on page 237..



```
<resource-ref>
     :
  <res-ref-name>jms/TwoPhaseQCF</res-ref-name>
  <res-type>javax.jms.QueueConnectionFactory</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
     :
<resource-ref>
     :
  <res-ref-name>jms/CommandQueue</res-ref-name>
  <res-type>javax.jms.Queue</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
```

Name	Type	J2EE Resource
jms/TwoPhaseQCF	javax.jms.QueueConnectionFactory	JMS2PhaseQCF
jms/CommandQueue	javax.jms.Queue	
		JMSQueueDest

Figure 12. Linking JMS connection factories and destinations to a WebSphere for z/OS configuration

The following list corresponds with the numbered items in Figure 12:

1. An application component's source code contains JNDI look-ups to find a JMS connection factory and destination. In the application component's deployment descriptor, the resource references in these JNDI look-ups are linked to specific resource types.
2. When you define the WebSphere for z/OS run-time environment for the application, you use the Administration application to:
   a. Define an RRS-enabled JMS ConnectionFactory and a JMS destination (queue) as J2EE resources associated with the J2EE server in which the application will run. By specifying properties for these resources, you link the logical name to MQSeries and a queue on z/OS or OS/390.
   b. Install the application that uses JMS, which includes resolving any resource references in the application's deployment descriptor. You resolve these resource references by mapping them to a JMS ConnectionFactory and destination that are defined to the J2EE server.

JavaMail™

JavaMail provides a framework for developing and supporting Java applications that send, store, and receive mail. According to the Sun Microsystems J2EE specification, a JavaMail configuration consists of the following:

- The JavaMail API implementation, which provides general facilities for reading and sending e-mail.
- The JavaBeans Activation Framework (JAF), another Java API that handles mail in forms that are more elaborate than plain text (in other words, MIMEs, URL pages, file attachments, and so on).
- Service providers, which implement protocols for mail transport and storage. In other words, these service providers allow applications to send mail through mail servers and to access stored mail. JavaMail currently includes three protocols:
  - Simple Mail Transfer Protocol (SMTP)
  - Internet Message Access Protocol (IMAP)
  - Post Office Protocol Version 3 (POP3)

Figure 13 illustrates a JavaMail configuration on z/OS or OS/390, with WebSphere for z/OS supporting almost all of the JavaMail elements enclosed in the bold rectangle. To have a functional mail system on z/OS or OS/390, your installation also needs to have the appropriate mail servers and mail stores installed.

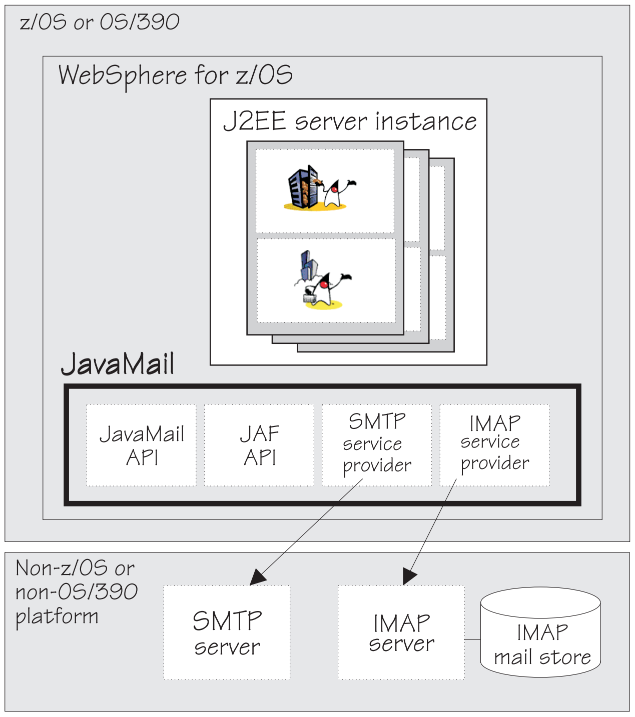


Figure 13. JavaMail and its supported elements in WebSphere for z/OS

The WebSphere for z/OS JavaMail package supports the use of the JavaMail API by all types of application components: Servlets, JavaServer Pages (JSPs), Enterprise JavaBeans, and application clients. This package contains:
- The JavaMail API implementation
- The JAF API
- Two service providers: An SMTP service provider and an IMAP service provider

Figure 14 on page 51 gives you a closer look at how these JavaMail elements are packaged in the WebSphere for z/OS run-time environment, for use by J2EE application components that are installed in a J2EE server. For installed applications to successfully use the JavaMail API, the J2EE server in which they run must have a JavaMail session defined as one of its J2EE resources.

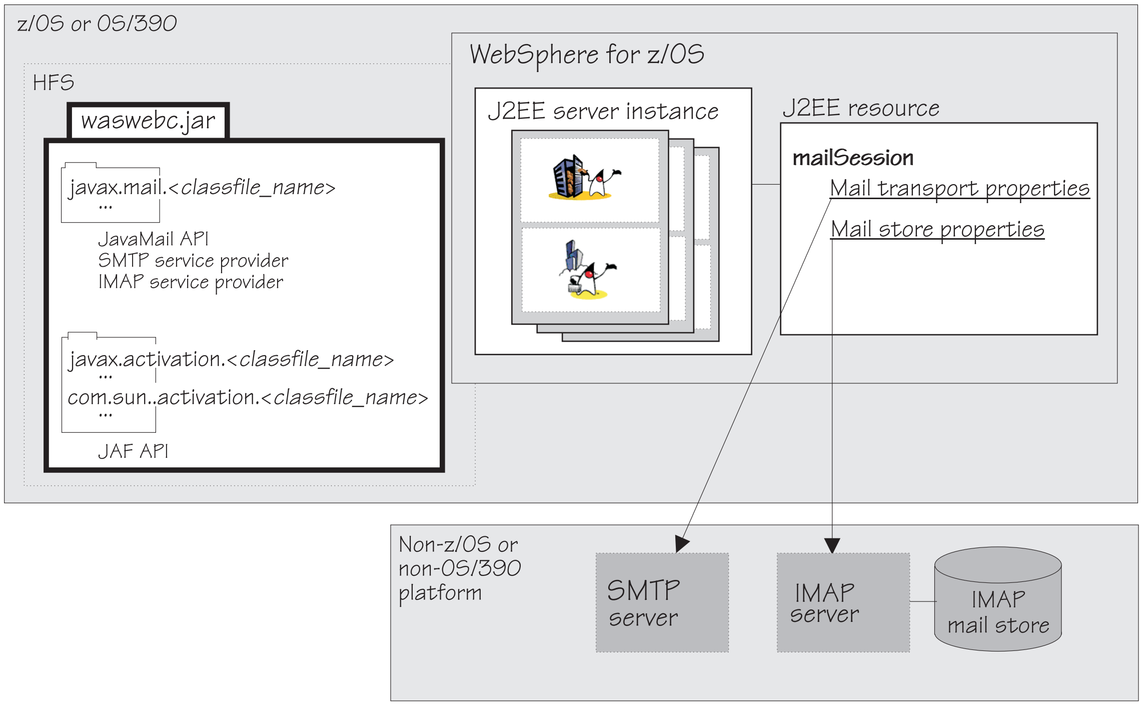


Figure 14. A closer look at the WebSphere for z/OS JavaMail package

To enable a JavaMail system with WebSphere for z/OS, you need to link the J2EE server’s mail session resource to the mail resource references that your application’s components use. Figure 15 on page 52 provides an overview of the steps required to complete this link; step-by-step instructions appear in Chapter 13, “Using the JavaMail API in J2EE application components,” on page 241.

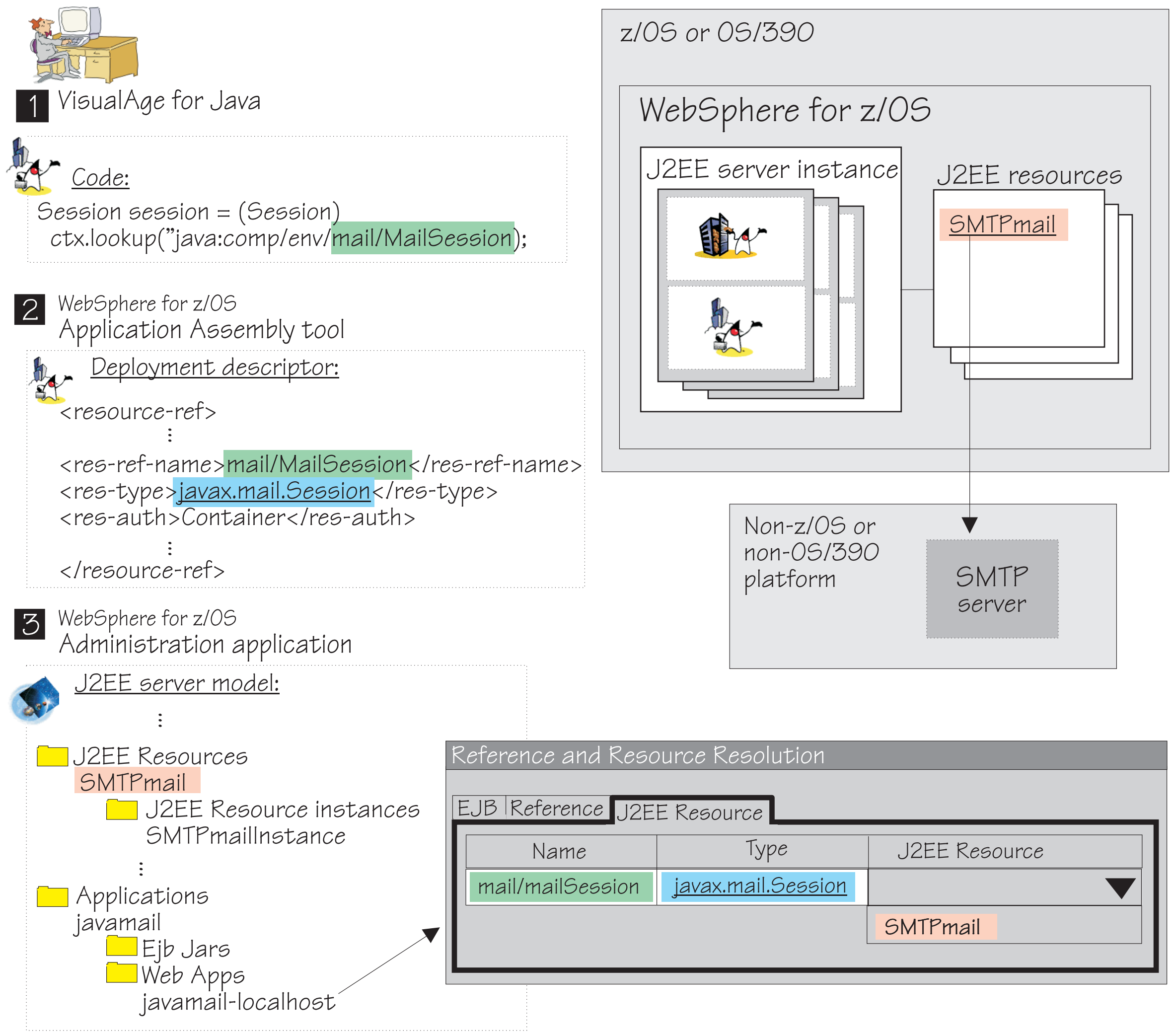


Figure 15. Enabling JavaMail for Web applications in WebSphere for z/OS

The following list corresponds with the numbered items in Figure 15:

1. In its source code, an application component uses a JNDI look-up to find a specific JavaMail session.

   Example:

   ```
   Session session = (Session) ctx.lookup("java:comp/env/mail/MailSession");
   ```

2. In the application component's deployment descriptor, the mail resource reference in the application code is linked to a specific resource type.

   Example:

   ```
   <resource-ref>
   <description>description</description>
   <res-ref-name>mail/MailSession</res-ref-name>
   <res-type>javax.mail.Session</res-type>
   <res-auth>Container</res-auth>
   </resource-ref>
   ```

3. In the J2EE server configuration, a JavaMail session is defined as a J2EE resource. When you install the application, you map the resource type in the application's deployment descriptor to the JavaMail session.
4. Once you activate the J2EE server, the application can then use this session to access the JavaMail implementation that WebSphere for z/OS provides. As long

as the z/OS or OS/390 system has a mail server and mail store installed, your application can create messages and to get store access.

WebSphere for z/OS also supports JavaMail's debugging capability, which allows you to collect diagnostic data for your application. If debugging is enabled for the J2EE server, this data appears in the stdout.log and stderr.log files. Debugging data looks like this:

```
DEBUG: getProvider() returning javax.mail.Provider[TRANSPORT,smtp]
DEBUG SMTP: useEhlo true, useAuth false

DEBUG: SMTPTransport trying to connect to host "smtp3.eseller.com", port 25

DEBUG SMTP RCVD: 220 relay14.eseller.com ESMTP Sendmail; Tue, 19 Dec 2000
15:08:42 -0700

DEBUG: SMTPTransport connected to host "smtp3.eseller.com", port: 25

DEBUG SMTP SENT: EHLO y2001
DEBUG SMTP RCVD: 250-relay14.eseller.com Hello testpc.eseller.com,
        pleased to meet you
250-8BITMIME
250-SIZE 20000000
250-DSN
250-ONEX
250-ETRN  250-XUSR
250 HELP

DEBUG SMTP SENT: MAIL FROM:<user1@mail.company.com>
DEBUG SMTP RCVD: 250 <user1@mail.company.com>... Sender ok
DEBUG SMTP SENT: RCPT TO:<user2@othermail.net>
DEBUG SMTP RCVD: 250 <user2@othermail.net>... Recipient ok
Verified Addresses    user2@othermail.net
DEBUG SMTP SENT: DATA
DEBUG SMTP RCVD: 354 Enter mail, end with "." on a line by itself

DEBUG SMTP SENT:
.
DEBUG SMTP RCVD: 250 PAA125654 Message accepted for delivery

DEBUG SMTP SENT: QUIT
```

For instructions for enabling JavaMail debugging, use both procedures in Chapter 13, "Using the JavaMail API in J2EE application components," on page 241..

IBM Extensions

The IBM deployment descriptor extensions provide additional, proprietary, policy meta-data for J2EE application components to be deployed on WebSphere servers. This extended policy meta-data allows you to exploit WebSphere qualities of service that are not addressed in the Sun Microsystem's J2EE specification for J2EE applications, Web applications, and enterprise beans. If you use these IBM extensions, you can still port the application components between WebSphere and non-WebSphere platforms, because the extensions do not change the semantic behavior of a J2EE application component.

To use the IBM extensions, add your J2EE application component to the WebSphere for z/OS Application Assembly tool. Then highlight the component, and click on one of the IBM Extensions tabs in the right side of the window. The following labels identify IBM Extensions tabs:
- RunAs
- ThreadID

- Isolation
- ReadOnly
- Policy
- Extensions

Table 7 lists only the extensions that introduce WebSphere for z/OS-specific considerations. If you plan to deploy an application component in a WebSphere for z/OS J2EE server, review the following summary for extensions that you can use, and read the associated information for each extension. Descriptions of all applicable IBM extensions are available in the help information for the Application Assembly tool.

Table 7. Summary of IBM deployment descriptor extensions

IBM extension	Applies for this type of J2EE application component:			Associated information
	Session bean	CMP entity bean	BMP entity bean	
Activate and **Load**	**Activate** only for stateful session beans	Yes	Yes	"Altering commit-time options"
Bean pool size	Yes	Yes	Yes	"Setting the size of a bean pool" on page 55
ReadOnly	No	Yes	No	"Optimizing end-of-transaction processing" on page 56
Isolation level	Yes	Yes	Yes	"Isolating transactions that access persistent data" on page 56
Pessimistic / Optimistic concurrency control	No	Yes	No	"Controlling concurrent access to persistent data" on page 57

Altering commit-time options

The IBM deployment descriptor extensions **Activate** and **Load** allow you to tailor EJB container behavior regarding commit time options that are defined in the EJB 1.1 specification. To alter these options, add your J2EE application component to the WebSphere for z/OS Application Assembly tool. Then highlight the component, and click on the `+Policy` tab in the right side of the window.

You may specify a combination of activate and load policy values for entity beans, but only activate policy values apply for stateful session beans.

Activate policy values:

Once
Specifies that a bean is activated only once in a given server region.

Note: When you select this value, WebSphere for z/OS attempts to honor the policy but, in accordance with the EJB specification, may passivate the bean if run-time conditions make it necessary to do so. This behavior means that selecting the activate once policy is only a performance optimization to reduce activation/passivation costs. Therefore, you must write appropriate logic for **both** `ejbActivate` and `ejbPassivate`.

AtTran
Specifies that a bean is activated at the start of each transaction, regardless of the transaction type (global or local).

Load policy values:

AtActivation
Specifies that a bean's essential state is loaded at the same time the bean is activated. This behavior applies for both CMP and BMP beans.

AtTran
Specifies that a bean's essential state is loaded at the start of each transaction, regardless of the transaction type (global or local).

Table 8. Activate/Load combinations for entity beans

Container behavior for corresponding commit time option from EJB 1.1 specification	Activate value for entity bean	Load value for entity bean
Option A: • Instance state is written to database. • Instance stays ready. • Instance has exclusive access to object state, so instance state remains valid. **Note:** "Exclusive access to object state" is the responsibility of the application provider; the WebSphere for z/OS J2EE server does not guarantee this condition.	Once	AtActivation
Option B: • Instance state is written to database. • Instance stays ready. • Instance does not have exclusive access to object state. Because instance state might not remain valid, the EJB container synchronizes instance state with persistent state at the beginning of each transaction.	Once	AtTran
Option C: • Instance state is written to database. • Instance does not stay ready; the EJB container returns the instance to the pool of available instances after the transaction completes. • Validity of instance state does not apply because instance does not stay ready.	AtTran	AtActivation or AtTran

Setting the size of a bean pool

The IBM deployment descriptor extension **Bean Pool Size** allows you to set, per Enterprise bean, the minimum and maximum sizes of the bean pool that the container will use for each bean type. To alter these options, add your J2EE application component to the WebSphere for z/OS Application Assembly tool. Then highlight the component, and click on the `+Extensions` tab in the right side of the window.

Each bean type (class) has a separately managed pool. The minimum pool size specifies the minimum number of beans to be left in the pool following an eviction cycle. The max pool size specifies the maximum number of beans allowed in the pool before the EJB container begins evicting beans from the pool.

Default values: Minimum size is 10; maximum is 250.

Note: Bean pool size is not relevant for `activate-once`, `load-once` beans. Such beans have exactly one instance and no more.

Optimizing end-of-transaction processing

The IBM deployment descriptor extension **ReadOnly** allows you to reduce the processing overhead at the end of a transaction. During the course of a transaction, if the only bean methods invoked are read-only methods, then the EJB container avoids the overhead of storing the bean's essential state back to the database. To achieve this optimization, specify the ReadOnly extension per method within a container-managed entity (CMP) bean.

Isolating transactions that access persistent data

The IBM deployment descriptor extension **Isolation level** allows you to set, per method defined on Enterprise bean remote or home interfaces, the type of locking that DB2 uses to protect data. Valid values are:

- `TRANSACTION_NONE` allows application assemblers to indicate that they do not want a specific JDBC isolation level to be enforced on connections acquired directly by the bean itself.
- `TRANSACTION_READ_UNCOMMITTED` allows methods within a transaction to read uncommitted changes made as part of a different transaction, before those changes are committed.
- `TRANSACTION_READ_COMMITTED` allows methods within a transaction to read only committed changes made as part of a different transaction. In other words, this isolation level prohibits "dirty reads."
- `TRANSACTION_REPEATABLE_READ` provides the same isolation as `TRANSACTION_READ_COMMITTED`, but also ensures that reading the same data multiple times returns the same value, even if other transactions modify the data. In other words, this isolation level prohibits "dirty reads" and "nonrepeatable reads."
- `TRANSACTION_SERIALIZABLE` provides the same isolation as `TRANSACTION_REPEATABLE_READ`, but also ensures that if a query retrieves a result set based on a predicate condition and another transaction inserts data that satisfies the predicate condition, rerunning the query returns the same JDBC result set. In other words, this isolation level prohibits "dirty reads," "nonrepeatable reads," and "phantom reads."

Rule: All methods that run under the same transaction must have the same isolation level (or the `TRANSACTION_NONE` default); otherwise, the EJB container throws `IsolationLevelChangeException`.

Default value: `TRANSACTION_NONE`

Note: For beans assembled and deployed with the default descriptor value `TRANSACTION_NONE`, WebSphere for z/OS actually uses the JDBC isolation level `TRANSACTION_REPEATABLE_READ` on connections that the EJB container acquires and uses to manage CMP persistent state in DB2.

WebSphere for z/OS uses the transaction isolation level as follows:

- For session beans and entity beans that use bean-managed persistence (BMP beans), the EJB container sets the isolation level when the bean requests a connection. If the bean explicitly sets the isolation level on the database connection, however, the container will cease to manage the isolation level setting on that connection. In other words, the connection isolation level becomes bean managed.
- For entity beans that use container-managed persistence (CMP beans), the container generates database access code that implements the specified isolation level. For beans assembled and deployed with the default descriptor value `TRANSACTION_NONE`, however, WebSphere for z/OS uses the JDBC isolation level

TRANSACTION_REPEATABLE_READ on connections that the EJB container acquires and uses to manage CMP persistent state in DB2.

For additional information about isolation level for CMP beans, see "Controlling concurrent access to persistent data."

Note: If you are porting BMP beans from WebSphere servers on the workstation to a WebSphere for z/OS J2EE server, you may need to change the SELECT ... FOR UPDATE statement to prevent DB2 from demoting update (U) locks to share (S) locks when the resultset returned by executing the SELECT statement is closed. Changes are required only for specific isolation levels, as follows:

If the BMP bean uses this isolation level:	Change the SELECT ... FOR UPDATE statement to:
TRANSACTION_REPEATABLE_READ	SELECT ... FOR UPDATE WITH **RS** KEEP UPDATE LOCKS
TRANSACTION_SERIALIZABLE	SELECT ... FOR UPDATE WITH **RR** KEEP UPDATE LOCKS

Controlling concurrent access to persistent data

For each Enterprise bean that uses container-managed persistence (CMP bean), you can use the IBM extension for concurrency control to determine how the WebSphere Application Server and relational database resource managers work together to handle multiple read or update access requests for the same data. For CMP beans that run in a WebSphere for z/OS J2EE server, the EJB container and DB2 are responsible for managing persistent data, and you can influence their behavior by selecting one of two commonly used approaches to concurrency control:

Pessimistic

This approach delegates all responsibility to the resource manager, DB2, which uses locking to provide concurrency control. The combination of two additional IBM deployment descriptor extensions, isolation level and ReadOnly method permission, determines which type of locking DB2 uses for each CMP bean method. The ReadOnly method permission identifies the access intent (read or update) for a specific method.

Optimistic

This approach requires the application to share responsibility for concurrency control, by using a programming technique that minimizes contention for database locks, and allows WebSphere for z/OS to check the consistency of data and determine whether to make the requested updates.

As for pessimistic concurrency control, the ReadOnly method permission identifies the access intent (read or update) for a specific method. In other words, the same end-of-transaction performance optimization is possible regardless of the concurrency model.

Each approach has advantages and disadvantages, so you need to understand your application's characteristics and requirements before you can determine the appropriate approach for each CMP bean. Base your choice for concurrency control on the information in Table 9 on page 58, which presents a summary of determining factors for each approach, along with recommendations and other notes. After you have decided which approach is appropriate for your application, use the WebSphere for z/OS Application Assembly tool to assemble the CMP beans. To make sure you understand the implications or your choice, and that you complete all of the appropriate assembly steps, read the following topics:

- "Checklist for using pessimistic concurrency control" on page 58

- “Checklist for using optimistic concurrency control” on page 61

If you also need general instructions for installing or using the WebSphere for z/OS Application Assembly tool, see Chapter 7, “Assembling a J2EE application,” on page 135.

Table 9. Deciding which concurrency control approach to use for CMP beans

If your application. . .	**Then. . .**	**Notes . . .**
Contains `cmp-fields` that map to only primary key and the following column types: • Binary large object (BLOB) • Character large object (CLOB) • `VARCHAR` with length greater than 255 • `LONG VARCHAR` • `float` or `double`	Use pessimistic concurrency control. **In this case, using the optimistic approach can result in updates being lost.**	**Notes:** 1. **Alternative:** In this case, you can use optimistic concurrency control safely only if you can add an eligible column type to the CMP mapping. For example, you could add an update counter to the mapping (and to the underlying database table) to increment each time an optimistic update is made. 2. **Restriction:** You cannot use **optimistic** concurrency control for Enterprise beans that have CMP fields of type `float` or `double` because there is a loss of precision when storing these fields into a DB2 database. This loss causes optimistic updates to fail.
Uses an access pattern that is predominantly read access	Use optimistic concurrency control.	In this case, using the optimistic approach can improve concurrency and throughput.
Uses an access pattern that is predominantly update access	Use pessimistic concurrency control.	In this case, the pessimistic approach might offer some advantage because update failures, if any, occur before processing begins for a given CMP bean. When failures occur before processing begins, the cost of recovery is relatively low. On the other hand, the optimistic approach also can be advantageous if concurrent updates to not collide.
Does not use a predominant access pattern	Consider the relative cost of recovering from an update failure.	With the pessimistic approach, update failures occur when locks cannot be obtained quickly enough (in other words, the failure occurs due to deadlock at time of read). With the optimistic approach, update failures occur when the underlying data changes after being read (in other words, the failure occurs at time of update). Because failures occur during, rather than before, processing begins for a CMP bean, these failures might require more difficult and costly recovery processing on the part of your application.

Checklist for using pessimistic concurrency control: Pessimistic concurrency control delegates all responsibility to the resource manager, DB2, which uses locking to provide concurrency control. The combination of two additional IBM deployment descriptor extensions, `isolation level` and `ReadOnly` method permission, determines which type of locking DB2 uses for each CMP transaction. Depending on the type of locking that DB2 uses, and the number and type of concurrent access requests, applications might have to wait for other processing to complete before their read or update operations can be made. Under these circumstances, applications might be affected not only by slower performance, but also by timing failures (that is, lock time-outs). Although your applications might be somewhat slower or possibly encounter more timing failures, pessimistic concurrency control is the better choice for applications with a relatively high number of update collisions (that is, transactions that update the same data fields).

With pessimistic concurrency control, WebSphere for z/OS also provides an integrity monitoring function through which it can detect the potential for either a deadlock or lost-update condition. For either condition, WebSphere for z/OS issues an informational message, only once per condition for each unique CMP bean running in a given J2EE server instance. These messages enable you to identify the following potential problems:
- Update transactions that start with a method marked with `ReadOnly` permission, and
- Update transactions that use an insufficient isolation level setting.

Unless you make other selections through the Application Assembly tool, CMP beans are assembled and deployed with the default isolation level of `TRANSACTION_NONE`, and the default method permission setting (which is equivalent to an access intent of update). If these defaults are appropriate for your application, you might not have to complete all of the assembly steps in the following checklist:

✓	**Item**
☐	Add your CMP bean and other application components, if any, to the latest version of the WebSphere for z/OS Application Assembly tool.
☐	Highlight the CMP bean, and click on the `+Extensions` tab in the right side of the window. Select `pessimistic concurrency control`.

✓	Item
☐	Verify the access intent for methods defined on the CMP bean's remote and home interfaces. The default access intent is update; if you need to change this default setting, select the `ReadOnly` method permission property. The setting for this property governs whether or not the SQL `SELECT` statement includes the `FOR UPDATE` clause for the first method in a transaction. (The first method in a transaction causes an entity bean's essential state to be loaded from the underlying resource manager.) • The `ReadOnly` setting is appropriate for methods that do not update the CMP bean's persistent state. In this case, the `FOR UPDATE` clause is not added to the `SELECT` operation, and DB2 will not obtain update locks. • The default setting (equivalent to update access intent) is appropriate for methods that update the CMP bean's persistent state. In this case, the `FOR UPDATE` clause is on the `SELECT` operation, and DB2 obtains update locks for entity beans configured for commit-time options B and C only. Entity beans configured for commit-time option A are assumed to have exclusive access to the database, so locking is not required. For additional information about commit-time options, see "Altering commit-time options" on page 54. **Rule:** Some applications use custom finders that contain the `FOR UPDATE` clause, or keywords `ORDER BY` and `DISTINCT`, on the `SELECT` operation. In these cases, you must use one of the following settings to avoid encountering an SQL error (`SQLCODE` -126) when the J2EE server attempts to run the CMP bean: • Specify the `ReadOnly` setting on the method level; • Specify env-entry `com.ibm.websphere.persistence.bean.managed.custom.finder.access.intent` as true in the bean's standard deployment descriptor (*ejb*`-jar.xml`); or • Specify the property `com.ibm.websphere.persistence.bean.managed.custom.finder.access.intent=1` in the JVM properties file for the J2EE server. Using this property in the J2EE server's JVM property file sets the same access intent for all CMP beans that run in the server. For additional information about this JVM property, see "JVM properties and properties files" on page 339. **Default:** update

✓	Item
☐	On a method level, highlight the CMP entity bean and select the `+Isolation` tab. If appropriate, modify the JDBC isolation level for methods defined on the CMP bean's remote and home interfaces. This IBM extension controls the type of locking that DB2 uses. Valid values are: • `TRANSACTION_NONE` • `TRANSACTION_READ_UNCOMMITTED` • `TRANSACTION_READ_COMMITTED` • `TRANSACTION_REPEATABLE_READ` • `TRANSACTION_SERIALIZABLE` For additional information about isolation level values, see "Isolating transactions that access persistent data" on page 56. **Default:** `TRANSACTION_NONE` **Note:** For beans assembled and deployed with the default descriptor value `TRANSACTION_NONE`, WebSphere for z/OS uses the JDBC isolation level `TRANSACTION_REPEATABLE_READ` on DB2 connections. **Rule:** You cannot specify different isolation levels for methods in the same transaction. If you do so, the EJB container throws an `IsolationLevelChangeException`. **Recommendation:** Unless the method is composed entirely of read operations, use an isolation level of `TRANSACTION_REPEATABLE_READ` or `TRANSACTION_SERIALIZABLE`. `TRANSACTION_REPEATABLE_READ` is recommended, whether the first method in the transaction is marked `ReadOnly` or not, because `TRANSACTION_REPEATABLE_READ` is the minimum level required for DB2 to hold locks across the load and store operations for an entity bean. If you choose a lower level of isolation for transactions that include update methods, you run the risk of potential lost updates. If a transaction starts with a `ReadOnly` method, but subsequently includes an update method, that transaction is exposed to a potential deadlock condition, if DB2 cannot obtain update locks because other transactions hold read locks on the same data. Data integrity is ensured, however, because WebSphere for z/OS rolls back the transaction.
☐	After completing property changes for your CMP bean, save your changes. If your application contains more than one CMP bean, make sure you check the IBM Extensions properties, as described above, for each of the remaining beans.
☐	Select the application in the tree view in the left pane, then select `Validate` to validate the contents of the application.
☐	After validation, select `Deploy` to deploy the application.
☐	After deployment, select `Export` to export the application.

Now you are ready to use the WebSphere for z/OS Administration application to install the application in a J2EE server.

Checklist for using optimistic concurrency control: Optimistic concurrency control requires the application to share responsibility for concurrency control, by using a programming technique that minimizes contention for database locks. Specifically, the application uses "over-qualified updates," which are update query predicates that compare previously saved values to current values, and perform updates based on the results of the comparison. This programming technique enables update and read operations to occur almost simultaneously, which results in improved transactional performance compared to the pessimistic approach to concurrency control. So optimistic concurrency control is the appropriate choice for applications for which performance is more important than avoiding update failures, and when the cost of recovering from update failures is relatively low.

To use over-qualified updates, your application performs as follows:

1. The application reads a number of table columns, in addition to the primary key, and saves the initial values for later comparison.
2. The application uses an SQL `UPDATE` that includes an SQL `WHERE` clause, using the saved values of the previously read table columns.

   For CMP beans, this type of update allows WebSphere for z/OS to check the consistency of data on behalf of the application, and determine whether to make the requested updates. WebSphere for z/OS creates a fully qualified update by forming a WHERE clause out of all eligible table columns, using only the following:
   - Fields defined on *cmp-field* elements within the CMP bean's deployment descriptor.
   - Column types. WebSphere for z/OS does not use the following column types:
     - Binary large object (BLOB)
     - Character large object (CLOB)
     - `VARCHAR` with length greater than 255
     - `LONG VARCHAR`

     If your application depends on ineligible column types in the WHERE clause, using the optimistic approach can result in updates being lost, unless you can use the alternative described in Table 9 on page 58.

     Restriction: You cannot use optimistic concurrency control for Enterprise beans that have CMP fields of type `float` or `double` because there is a loss of precision when storing these fields into a DB2 database. The loss of precision occurs because Java floating point is IEEE format, whereas DB2 stores only zSeries Hex floating point. While DB2 performs conversions to or from IEEE and Hex floating point format during database load or store operations, an inherent round-off error occurs. The resulting change in precision interferes with the correct operation of the "over-qualified update" programming technique used for optimistic concurrency control.

     Recommendation: Use decimal types instead of floating point, whenever possible.

   If WebSphere for z/OS detects that the values have changed since the application last read and saved the values, it does not perform the update; instead, the query results in an "optimistic update failure" and WebSphere for z/OS rolls back the transaction.
3. The application is designed to recover from optimistic update failures. In such cases, WebSphere for z/OS returns the following exceptions:
   - For container-managed transactions, the application client receives a `RemoteException`.
   - For bean-managed transactions, the application client receives a `TransactionRolledBackException`.

Unless you make other selections through the Application Assembly tool, CMP beans are assembled and deployed with the default isolation level of `TRANSACTION_NONE`, and the default method permission setting (which is equivalent to an access intent of update). To modify these defaults and correctly use optimistic concurrency control, complete the steps in the following checklist:

✓	**Item**
☐	Make sure that your application is designed to use "over-qualified updates," as described above.
☐	Add your CMP bean and other application components, if any, to the latest version of the WebSphere for z/OS Application Assembly tool.
☐	Highlight the CMP bean, and click on the `+Extensions` tab in the right side of the window. Select `optimistic concurrency control`. **Rule:** You cannot use optimistic concurrency control with commit-time option A. For additional information about commit-time options, see "Altering commit-time options" on page 54.
☐	Verify the access intent for methods defined on the CMP bean's remote and home interfaces. The default access intent is update; if you need to change this default setting, select the `ReadOnly` method permission property. In contrast to pessimistic concurrency control, access intent for optimistic control does not determine whether the `FOR UPDATE` clause is added to the SQL `SELECT` for the first method in a transaction. For CMP beans using optimistic control, the SQL `SELECT` statement never contains a `FOR UPDATE` clause. • The `ReadOnly` setting is appropriate for methods that do not update the CMP bean's persistent state. • The default setting (equivalent to update access intent) is appropriate for methods that update the CMP bean's persistent state. **Default:** update
☐	On a method level, highlight the CMP entity bean and select the `+Isolation` tab. If appropriate, modify the JDBC isolation level for methods defined on the CMP bean's remote and home interfaces. This IBM extension controls the type of locking that DB2 uses. Valid values are: • `TRANSACTION_NONE` • `TRANSACTION_READ_UNCOMMITTED` • `TRANSACTION_READ_COMMITTED` • `TRANSACTION_REPEATABLE_READ` • `TRANSACTION_SERIALIZABLE` For additional information about isolation level values, see "Isolating transactions that access persistent data" on page 56. **Default:** `TRANSACTION_NONE` **Rule:** You cannot specify different isolation levels for methods in the same transaction. If you do so, the EJB container throws an `IsolationLevelChangeException`. **Recommendation:** Use an isolation level of `TRANSACTION_READ_COMMITTED`, for which DB2 will hold locks only during `READ` and `UPDATE` operations themselves, rather than across the `READ/UPDATE` period. You should not use `TRANSACTION_REPEATABLE_READ` or `TRANSACTION_SERIALIZABLE` because DB2 will acquire and hold locks for the life of the transaction, which defeats the purpose of using optimistic concurrency control. You may use lower isolation levels only for transactions that contain `READ` operations.
☐	After completing property changes for your CMP bean, save your changes. If your application contains more than one CMP bean, make sure you check the IBM Extensions properties, as described above, for each of the remaining beans.
☐	Select the application in the tree view in the left pane, then select `Validate` to validate the contents of the application.

✓	**Item**
☐	After validation, select `Deploy` to deploy the application.
☐	After deployment, select `Export` to export the application.

Now you are ready to use the WebSphere for z/OS Administration application to install the application in a J2EE server.

Connectors

If you want to develop J2EE application components that access legacy transactions and data under CICS or IMS, you may use one or more of the CICS and IMS connectors that WebSphere for z/OS currently supports.

WebSphere for z/OS supports the following CICS or IMS connectors, which are designed to use the Sun Microsystems Corporation's Java 2 Platform, Enterprise Edition (J2EE) Connector Architecture:
- CICS Transaction Gateway External Call Interface (ECI) Connector
- IMS Connector for Java
- IMS JDBC Connector

These connectors, which are also known as resource adaptors, not only implement the J2EE connector interfaces but also are RRS-compliant; in other words, they are designed specifically to work with the resource recovery services (RRS) component of z/OS or OS/390. Resource recovery consists of the protocols and program interfaces that allow WebSphere for z/OS, the RRS component of z/OS or OS/390, and CICS or IMS to work together to make consistent changes to multiple protected resources. Protected resources are considered so critical to a company's work that the integrity of these resources must be guaranteed.

Because of their design, WebSphere for z/OS, the RRS component of z/OS or OS/390, CICS or IMS subsystems and these RRS-compliant connectors can participate in two-phase commit processing, which enables z/OS or OS/390 to restore critical resources to their original state if they become corrupted because of a hardware or software failure, human error, or a catastrophe. These J2EE connectors are shipped as part of separate CICS or IMS products, and are considered the strategic connectors for connecting to CICS and IMS.

For its supported connectors, WebSphere for z/OS also provides additional advantages:
- The ability for system administrators to define connection management at a sysplex level, so that all WebSphere for z/OS J2EE servers benefit from efficient use of the system resources associated with connections. Connection management support is a configuration extension available through the WebSphere for z/OS Administration application.
- The ability for application assemblers to specify:
  - Connection management policy, which is a quality of service issue for applications using connectors. This ability allows finer control of the management of valuable back-end resources, which is especially useful to prevent a misbehaving application from tying up system-wide resources, thereby making the system unusable.
  - Resource authentication for applications using connectors. This ability determines which user identities WebSphere for z/OS will pass to back-end products (such as CICS and IMS) through connectors.

Connection management policies and resource authorization are set through the WebSphere for z/OS Application Assembly tool.

These configuration and application extensions are functions that WebSphere for z/OS provides in addition to the implementation of the J2EE interfaces. Use of these extensions does not cause any loss of function provided for J2EE compliance at the current level.

WebSphere for z/OS also extends its connection management capabilities to its JDBC resources, so J2EE application components that use JDBC to access DB2 also benefit from additional qualities of service. Although WebSphere for z/OS treats DB2 JDBC datasources as managed connections, it does not treat DB2 JDBC connections exactly the same as CICS and IMS managed connections. For example, WebSphere for z/OS enforces resource authentication and connection reuse for DB2 JDBC connections, even when connection management support is not explicitly selected through the WebSphere for z/OS Administration application. When these differences affect how your installation uses a specific connector, further details are provided in the appropriate procedures.

WebSphere Application Server V4.0.1 for z/OS and OS/390 also provides "beta" CICSEXCI and IMSAPPC connectors, which are available as a download at:

`http://www.ibm.com/software/webservers/appserv/zos_os390/support.html`

WebSphere for z/OS also treats these connectors as managed connections; these connectors also are J2EE-compliant and RRS-compliant.

WebSphere for z/OS also supports the use of Common Connector Framework (CCF) connectors by Web components (servlets) only. This support is equivalent to the same level of support provided with previous versions of WebSphere for z/OS, and is intended as a migration aid for existing Standard Edition customers. IBM recommends moving to WebSphere for z/OS-supported connectors that are designed to implement the Sun Microsystems Corporation's J2EE Connector Architecture.

As specified by the J2EE Connector Architecture:

- A connector is responsible for sending and receiving data to and from a back-end resource, such as CICS or IMS.
- The application server is responsible for managing physical connections, or `ManagedConnections`, to a back-end resource, and for providing qualities of service related to the use of connectors. These qualities of service include connection pooling, transaction management, and security management. In a WebSphere for z/OS J2EE server configuration, these `ManagedConnections` are known as J2EE resources.

Given this specification, J2EE application components do not require knowledge of specific connector implementations; components deal only with application-level handles (known as `Connections`) to the J2EE server resources. Before installing J2EE application components, however, application assemblers and deployers might want to make some changes to deployment descriptors, to exploit the qualities of service available through a WebSphere for z/OS J2EE server and its supported connectors.

The following topics describe topics to review before you start using the WebSphere for z/OS connectors. The topics include assembly and deployment considerations for J2EE application components (Enterprise beans, servlets and JavaServer Pages) that you plan to install in a WebSphere for z/OS J2EE server.

Topic or subtask	Associated information (See . . .)
Deciding which connector to use	"Deciding which connector to use"
Guidelines for using connectors	• "Guidelines for accessing legacy programs" on page 68 • "Coding connector lookups" on page 68 • "Connector transaction processing" on page 70
Using connection management for more efficient use of connection resources	"Exploiting connection management support" on page 71
Determining the user ID that WebSphere for z/OS uses to check authority to access a connector	"Determining the user ID for resource authentication" on page 73
Checklist for creating an application component that uses connections	"Checklist for application components that use connectors" on page 77
Configuring the J2EE server, connector, and subsystem for back-end resources, and installing application components	"Configuring the WebSphere for z/OS-supported connectors" in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834

Deciding which connector to use

Table 10 lists the J2EE connectors that you may use to access CICS or IMS resources. Your installation must adhere to the configuration requirements stated in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834; any attempt to use these connectors in alternative configurations is not supported.

Note: *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834 contains additional information about determining which connector to use, based on the requirements of your J2EE application components or the network configuration at your installation.

Table 10. Deciding which connector to use

For these application requirements:	Use the following connector:	Guidelines and notes:
Access to CICS CommArea-based transaction programs	CICS Transaction Gateway ECI Connector	To use the CICS Transaction Gateway ECI Connector, CICS, CICS Transaction Gateway, and the WebSphere for z/OS J2EE server in which your component is installed must all run on the same z/OS or OS/390 system. If you currently are using the CICS Common Connector Framework (CCF) connector, which is not a J2EE connector, you should rework your application components to use this new CICS ECI connector, as soon as you can.

Table 10. Deciding which connector to use (continued)

For these application requirements:	Use the following connector:	Guidelines and notes:
Access to IMS transaction programs	IMS Connector for Java	To use the IMS Connector for Java and exploit the benefits of resource recovery processing, IMS, IMS Connect, and the WebSphere for z/OS J2EE server in which your component is installed must all run on the same z/OS or OS/390 system. If your application requires access to IMS resources on a remote z/OS or OS/390 system, you may use either the "beta" IMSAPPC connector (to exploit resource recovery processing) or the remote TCP/IP configuration for the IMS Connector for Java. In the remote TCP/IP configuration, the IMS Connector for Java runs as a non-transactional connector and, as such, does not provide resource recovery protection for your applications or data. If you currently are using the IMS Common Connector Framework (CCF) connector, which is not a J2EE connector, you should rework your application components to use this new IMS connector, as soon as you can.
	The "beta" IMSAPPC connector	If you require access to IMS resources on a remote z/OS or OS/390 system, you may use the IMSAPPC connector, which implements the J2EE Connector Architecture. APPC is another z/OS or OS/390 component that is designed to work with RRS using the two-phase commit protocol, so using this connector provides support for resource recovery that is equivalent to the support provided through the strategic connectors. For configuration requirements and procedures, see the documentation for the "beta" connectors, which is available through the WebSphere Application Server Web page: `http://www.ibm.com/software/webservers/appserv/` **Alternative:** If you require access to IMS resources on a remote z/OS or OS/390 system and you must use the TCP/IP protocol rather than APPC, you may use the IMS Connector for Java to access IMS on remote systems. Note, however, that this connector configuration does not provide resource recovery protection.
Access to IMS databases	IMS JDBC Connector	On z/OS or OS/390, using JDBC

Datasource lookup with backwards compatibility with Version 3.5

In order to provide compatability for servlets developed to run in a WebSphere Application Server Standard Edition Version 3.5 environment, V4.01 allows you to use the Version 3.5 programming model to look up a datasource in JNDI. In contrast to the V4.0.1 programming model that uses a J2EE style of looking up resources within an application-specific JNDI namespace, the Version 3.5 programming model features a pre-J2EE style of datasource lookup in a global JNDI namespace.

The Version 3.5 pre-J2EE style code looks like the following:

```
Context ctx = ...;
DataSource ds =(DataSource)ctx.lookup("jdbc/myDataSource");
```

The Version 4.0.1 J2EE style code looks like the following:

```
Context ctx = ...;
DataSource ds =(DataSource)ctx.lookup("java:comp/env/jdbc/myDataSource");
```

Furthermore, a Version 4.0.1 application is assembled along with a resource reference deployment descriptor, which is resolved in the adminstrative console by binding this resource reference to a physical datasource definition. The Version 3.5 application is not assembled with a datasource resource reference.

In a Version 4.0.1 environment, the Version 3.5 pre-J2EE style datasource lookup works with the following limitations:

- The DB2 datasource resource must be defined in the adminstrative console with a resource name that matches the string used in the application JNDI lookup invocation (minus the ″jdbc/″ prefix).

  For example, if the application code does a ctx.lookup(″jdbc/myDataSource″), then a resource of J2EE Resource Type ″DB2datasource″ must be defined in the adminstrative console with the Resource Name of myDataSource. (Note that it is not the Resource Instance name that is important.)
- The datasource will be managed by WebSphere Application Server Connection Management as if it was defined with a ″Connection Management Policy″ setting of ″Normal″ and a ″Resource Authentication″ setting of ″Application″. Because no resource reference is created for the application's datasource, the assembler does not have an opportunity to choose the previous two settings.

DB2 resource interactions initiated using a Version 3.5 pre-J2EE style datasource lookup will still be able to fully participate in global transactions.

Guidelines for accessing legacy programs

IBM recommends using a stateless session Enterprise bean to represent the legacy CICS or IMS program (in other words, use the bean as a wrapper for the target CICS or IMS program). This approach offers the following advantages:

- Enterprise beans have policy attributes that can be set during assembly and deployment; these attributes enable application assemblers and deployers to have more control over how WebSphere for z/OS processes requests driven to the stateless session bean.
- Enterprise beans are reusable. Client business applications on remote systems and J2EE application components installed in WebSphere for z/OS J2EE servers can drive requests through the stateless session bean to connect to back-end CICS and IMS processing.

Note: In WebSphere for z/OS, servlets and JSPs are dispatched under a stateless session Enterprise bean, so you can design them to use a connector to access applications and data under CICS and IMS. Although this design is possible, IBM recommends using an Enterprise bean to connect to back-end IMS and CICS processing, rather than designing servlets to obtain direct connections, because of the advantages listed above.

Coding connector lookups

Because WebSphere for z/OS uses the Java Naming and Directory Interface (JNDI) for lookup processing, J2EE application components can be designed to:

- Associate a logical name with a ConnectionFactory of a particular type (for example, MyConnectionFactory), and
- Use that logical name to do a lookup of the factory in the `java:comp/env` name space associated with the application component.

At run-time under a WebSphere for z/OS J2EE server, when the application invokes JNDI lookup processing to locate the ConnectionFactory it needs, the lookup request returns a ConnectionFactory associated with a specific Enterprise Information System (EIS) configuration (for example, a specific IMS subsystem and connector defined in the J2EE server configuration).

To use this lookup support:

1. The application component provider writes code to do a lookup of a ConnectionFactory.

   Example:

   ```
   // Obtain the initial JNDI context

   Context initctx = new InitialContext();

   // Perform JNDI lookup to obtain connection factory

     javax.resource.cci.ConnectionFactory cf =
    (javax.resource.cci.ConnectionFactory) initctx.lookup("java:comp/
    env/MyCICSECIConnectionFactory");
   ```
2. Application assemblers must ensure that a ConnectionFactory reference name and type are included in the list of resources the application component uses. For example, to access CICS resources, you use a name like `MyCICSECIConnectionFactory` and the type `javax.resource.cci.ConnectionFactory`. Application assemblers usea WebSphere application assembly tool to identify these reference names and types. "A WebSphere application assembly tool" means either the z/OS edition of the Application Assembly tool, the WebSphere Studio Application Developer tool, the WebSphere Studio Application Developer Integration Edition tool, or the Direct Deployment Tool/390fy.
3. Application installers must associate the reference name and type used in the application component to a specific J2EE resource defined in the J2EE server configuration. For example, using the WebSphere for z/OS Administration application, the installer:
   a. Defines a J2EE resource for a CICS subsystem, specifying `CICS_ECIConnectionFactory` as the type of resource, along with other properties that identify the CICS subsystem.
   b. Installs the application component. During this process, the installer matches the component's resource reference, `MyCICSECIConnectionFactory`, and its generic type, `javax.resource.cci.ConnectionFactory`, to the name of the J2EE resource defined as type `CICS_ECIConnectionFactory`.

   At the end of this installation process, after the J2EE server is activated, the logical name `MyCICSECIConnectionFactory` is added to the `java:comp/env` name space, and the ConnectionFactory is configured to a specific instance of the CICS subsystem and connector.

If an Enterprise bean does the lookup for a ConnectionFactory itself, lookup processing should be performed as shown in the above example. On the other hand, if a command bean is used by an Enterprise bean to encapsulate the connector support, the command bean will do the lookup. Likewise, when you are using WebSphere Studio Application Developer Integration Edition to develop an application, the connector lookup also will be encapsulated by the tooling and will not need to be coded by the developer.

Note: All instances of a given Enterprise bean class that do a lookup for the same ConnectionFactory may be returned the same ConnectionFactory instance. In this case, the ConnectionFactory may be a shared resource. Because of this possibility, IBM recommends that the ConnectionFactory not be modified during the course of processing by instances of the same Enterprise bean class.

Connector transaction processing

WebSphere for z/OS supports only two types of connectors: non-transactional and RRS-transactional. Connector transaction processing varies for each type, as described below.

Restriction: WebSphere for z/OS does not support XA transaction support or local transaction support defined by the J2EE Connector Architecture.

RRS-transactional

This type of connector is configured to work with the resource recovery services (RRS) component of z/OS or OS/390 to participate in two-phase commit processing. For RRS-transactional connectors, the type transaction processing performed is determined at the time an interaction is executed on a connection to send a request to the target Enterprise Information System (EIS). There are two ways a given interaction may be handled:

1. If processing under the current thread is running under a global transaction, WebSphere for z/OS propagates the current transaction context across the interface to the back-end EIS, and two-phase commit processing or rollback processing of the transaction will be coordinated using z/OS or OS/390 resource recovery services (RRS).
2. If processing under the current thread is not running under a global transaction, WebSphere for z/OS sends the request to the back-end EIS, indicating that processing performed for the request should be committed before returning (this type of processing is known as sync-on-return).

The transaction policy of the Enterprise bean (under which the connector support runs) dictates whether or not processing is running under a global transaction. If the transaction policy dictates processing under a global transaction, then any connector processing will also do global transaction processing. Similarly, if the transaction policy dictates processing without a transaction, then any connector processing will be performed as a sync-on-return request.

For example, suppose you use a stateless session Enterprise bean to represent a CICS program, and drive requests to the bean to get a connection to CICS and access the program. If you deploy the bean with a transaction policy of TX_REQUIRED, all processing for the bean runs under a global transaction. On the other hand, if you deploy the bean with a transaction policy of TX_SUPPORTS, bean processing runs under the transaction state of its caller. In this case, processing could be done under a global transaction for one caller, and processing for a different caller could be performed as a sync-on-return request.

Non-transactional

In the case of a connector that has been configured as a non-transactional connector, all requests to the back-end EIS (for example, a CICS or IMS subsystem) are performed as sync-on-return requests. In other words, any changes made by the EIS are committed by the time control is returned to the Enterprise bean that made the request.

Sync-on-return processing is performed regardless of the transaction policy specified by the Enterprise bean.

Exploiting connection management support

WebSphere for z/OS provides connection management policies that define how the J2EE server manages the connection resources that J2EE application components acquire when using a J2EE connector. To exploit connection management support, connection management must be selected using the WebSphere for z/OS Administration application.

Application assemblers or deployers set specific policy values through the J2EE application components' deployment descriptor, using the WebSphere for z/OS Application Assembly tool. From the policy set in the deployment descriptor, the WebSphere for z/OS J2EE server assigns a policy value to resources obtained from a connection factory. The J2EE server uses this policy for all connections.

The following topics describe the connection management policy values, their meaning, and the qualities of service that the WebSphere for z/OS J2EE server can provide as part of its connection management support.

Connection management policies

Using the WebSphere for z/OS Application Assembly tool, the application assembler may define one of the following connection management policy values in the deployment descriptor for a J2EE application component. The policy value dictates how the J2EE server in which the component is installed will manage connections:

Normal
: The J2EE server polices the J2EE application component's usage of connections, and when component instances are removed from the J2EE server cache, any connections owned by that component are reclaimed by the J2EE server.

Aggressive
: The J2EE server polices the J2EE application component's usage of connections, and is used as a trigger for component instance cache management. Components that cache connections from connection factories configured with an aggressive management policy may be evicted from the server's active cache. During the eviction process, the component is driven through the component framework methods (for example, unsetEntityContext) and allows the component to return the aggressively managed resource. After the eviction, the J2EE server reclaims any connection that has not been returned.

Default
: The J2EE server assigns a default to connection factories based on platform-specific issues. In particular, the WebSphere for z/OS J2EE server assigns a default policy of aggressive for JDBC connection factories, and a default policy of normal for all other connection factories.

The decision to select aggressive instead of normal policy means that limiting the number of critical resources the J2EE server allows components to consume is more important than cache efficiency. The enforcement of this policy is invisible to the programming model of the J2EE application components; however, J2EE server performance might be affected.

Connection processing models

Based on the Java Connector Architecture, Enterprise beans may use J2EE connectors in two ways:

1. One way is to get a connection, use it, and then close it within the lifecycle of a transaction. This way is referred to as the Get/Use/Close model.
2. The second way is for a stateful Enterprise bean to get a connection when it is initially created, and then save the connection for use by subsequent method invocations. This way is referred to as the Caching model.

WebSphere for z/OS connection management supports the Get/Use/Close model. The Caching model, however, is not supported in that each time a method is invoked, WebSphere for z/OS does not reassociate the cached connection handle with a ManagedConnection that has the new caller's security credentials. Instead, the cached connection is always associated with the ManagedConnection for the user under which the connection was obtained.

Installations that wish to implement Enterprise beans that use a cached connection may do so, but should be aware that the user that the connection is associated with may change in the future when WebSphere for z/OS supports the use of the ManagedConnection associateConnection() interface to reassociate a connection handle.

Connection sharing

WebSphere for z/OS does not support connection sharing. This restriction means that different Enterprise beans cannot share the same connection; for example, suppose:

- Enterprise bean A gets a connection.
- While running under the same transaction, Bean A invokes Enterprise bean B.
- Bean B also gets a connection.

In the above example, the connection obtained by Bean B is associated with a new ManagedConnection, rather than sharing the same ManagedConnection with which Bean A's connection is associated.

Connection pooling and reuse

To minimize the processing required each time a J2EE application component requests a connection, WebSphere for z/OS uses a connection pool for resources that the J2EE server needs to manage a connection. These resources include objects that the components use to refer to the connection, objects that the server uses to maintain information about the managed connection, and physical connections to subsystems such as DB2.

You can further reduce overhead, and avoid potential problems with limited system resources, by enabling connection reuse. To do so, set the JVM property `com.ibm.ws390.ConnectionUsageScopeDefault` for the J2EE server.

Recommendation: Setting this JVM property is particularly useful for J2EE application components that, within a single global transaction, repeatedly get, use, and close a DB2 JDBC connection. The default behavior for the J2EE server is to use a new physical connection until the global transaction is committed or rolled back, so these repeated get/use/close operations might exhaust the limited supply of physical DB2 connections. When you set the JVM property to the value `SeriallyReusable`, however, the J2EE server creates a connection pool that is associated with the global transaction, and returns connections to the pool whenever the J2EE application component closes the connection. The returned connection is then available for reuse within the transaction.

When you set the JVM property `com.ibm.ws390.ConnectionUsageScopeDefault` to the value to `SeriallyReusable`, the J2EE server can reuse connections to any J2EE connector that is defined to the server configuration. For instructions on setting the JVM property, see "JVM properties and properties files" on page 339.

Note: Connection pooling and reuse support works for DB2 JDBC connections even when connection management is not selected for J2EE servers, through the WebSphere for z/OS Administration application.

Connection monitoring and clean-up

WebSphere for z/OS does not allow resources to be retained after the passivation or removal of an Enterprise bean. If an Enterprise bean gets a connection but fails to close the connection across lifecycle boundaries, the J2EE server reclaims the connection during post-invoke processing for the Enterprise bean.

Determining the user ID for resource authentication

The J2EE Connector Architecture specifies that either the container or a J2EE application component can control the identity that a J2EE server uses to authenticate requests for a J2EE resource. A combination of application code, deployment descriptor values, and run-time environment settings determines which user identity is associated with the connection to a J2EE resource; this user ID then may be used for resource authentication. The following list briefly summarizes the values and settings that determine the user ID to be associated with a connection that is explicitly obtained through the `getConnection` method in the application component's code:

- Application programmers may specify a particular user identity and password on the `getConnection` method. These method parameters are optional, however, so values do not have to be set in the application component's code.
- Application assemblers or deployers use the `res-auth` deployment descriptor element to specify whether the container or application component will determine the user ID. They also may set or alter the user ID through the `RunAs` deployment descriptor element.

  Application assemblers use the WebSphere for z/OS Application Assembly tool to set this element in the deployment descriptor.
- Application installers can configure a WebSphere for z/OS J2EE server to honor an application's deployment descriptor settings for `RunAs` and for container-managed authentication. To do so, they use the Administration application to select the `Enable Setting OS thread ID to RunAs ID` property.

To correctly establish a specific user ID to be associated with a connection, use Table 11 on page 74 to select the desired outcome and learn which assembly and configuration tasks are required for each option. This information is based on the assumption that you cannot or do not want to alter application source code (that is, you do not want to add or remove specific user ID values on the `getConnection` method).

Notes:

1. The information in this table applies only to connections that are explicitly obtained through the `getConnection` method in the application component's

code. The identity used for connections that WebSphere for z/OS obtains for its own use (to manage persistent state for a CMP bean, for example) are determined as follows:

- If the application component is assembled with `RunAs` set to `caller`, and the J2EE server is configured with `Enable setting OS thread ID to RunAs ID` selected, WebSphere for z/OS associates the connection with the caller's user ID.
- Otherwise, WebSphere for z/OS associates the connection with the user ID of the J2EE server region.

2. For further information about run-time configuration tasks, see the following topics:
   - To determine what authority is required for the user ID associated with a connection, based on the resource to be accessed, see Table 2 on page 24.
   - For step-by-step instructions for defining connectors as J2EE resources associated with a J2EE server, see *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834.

Table 11. Assembly and configuration tasks for setting the identity for a J2EE resource connection

If you want this user ID associated with a connection:	**Complete these application assembly and run-time configuration tasks:**
User ID and password values specified on the `getConnection` method	**Assembly:** Select the appropriate value for the `res-auth` element in the deployment descriptor: • `Application` for Enterprise beans • `Servlet` for servlets **Note:** WebSphere for z/OS ignores the settings for the `RunAs` and `Set OS thread identity to RunAs identity` deployment descriptor elements. **Configuration:** Make sure that the `getConnection` user ID has appropriate authorization for the resource.

Table 11. Assembly and configuration tasks for setting the identity for a J2EE resource connection (continued)

If you want this user ID associated with a connection:		**Complete these application assembly and run-time configuration tasks:**
User ID of the J2EE server region	If the `getConnection` method **does not** specify a user ID	**Assembly:** Select one of the following options: 1. Select the value `container` for the `res-auth` element, and select the value `server` for the `RunAs` element. **Note:** If you choose this option, you do not have to select the `Set OS thread identity to RunAs identity` element in the deployment descriptor. If you do select the `Set OS thread identity to RunAs identity`, however, you **must** also select the J2EE server property `Enable Setting OS thread ID to RunAs ID` during configuration. Otherwise, the `getConnection` method will return a `No_permission` failure during run-time. 2. Select the value `application` for the `res-auth` element. **Configuration:** 1. Make sure that the server region ID has appropriate authorization for the resource. 2. If you are using the CICS Transaction Gateway ECI connector or the IMS Connector for Java, specify the server region user ID and password when you define the connector as a J2EE resource instance. **Note:** The server region user ID and password are implicit defaults for the DB2 JDBC and "beta" IMSAPPC connections.
	If the `getConnection` method **does** specify a user ID, override that value	**Assembly:** Select the value `container` for the `res-auth` element, and select the value `server` for the `RunAs` element. **Note:** In this case, you do not have to select the `Set OS thread identity to RunAs identity` element in the deployment descriptor. If you do select the `Set OS thread identity to RunAs identity`, however, you **must** also select the J2EE server property `Enable Setting OS thread ID to RunAs ID` during configuration. Otherwise, the `getConnection` method will return a `No_permission` failure during run-time. **Configuration:** Make sure that the server region ID has appropriate authorization for the resource.
User ID of the caller		**Assembly:** 1. Select the value `container` for the `res-auth` element in the deployment descriptor 2. Select the value `caller` for the `RunAs` element (which is the same as no explicit setting for `RunAs`) **Note:** If you choose this option, you do not have to select the `Set OS thread identity to RunAs identity` element in the deployment descriptor. If you do select the `Set OS thread identity to RunAs identity`, however, you **must** also select the J2EE server property `Enable Setting OS thread ID to RunAs ID` during configuration. Otherwise, the `getConnection` method will return a `No_permission` failure during run-time. **Configuration:** 1. Make sure that the caller's user ID has appropriate authorization for the resource. 2. If your application is requesting a DB2 JDBC connection, you must make sure that the J2EE server property `Enable Setting OS thread ID to RunAs ID` is selected. Otherwise, the DB2 connections will be associated with the user ID of the J2EE server region. For connections to other J2EE resources such as CICS and IMS, WebSphere for z/OS uses the RunAs user ID currently associated with the thread, regardless of the J2EE server property setting.

Table 11. Assembly and configuration tasks for setting the identity for a J2EE resource connection (continued)

If you want this user ID associated with a connection:	Complete these application assembly and run-time configuration tasks:
RACF user ID mapped to a security role	**Assembly:** 1. Select the value `container` for the `res-auth` element in the deployment descriptor 2. Specify the security role name as the value for the `RunAs` element **Note:** If you choose this option, you do not have to select the `Set OS thread identity to RunAs identity` element in the deployment descriptor. If you do select the `Set OS thread identity to RunAs identity`, however, you **must** also select the J2EE server property `Enable Setting OS thread ID to RunAs ID` during configuration. Otherwise, the `getConnection` method will return a `No_permission` failure during run-time. **Configuration:** 1. Make sure that the RACF user ID has appropriate authorization for the resource. 2. If your application is requesting a DB2 JDBC connection, you **must** make sure that the J2EE server property `Enable Setting OS thread ID to RunAs ID` is selected. Otherwise, the DB2 connections will be associated with the user ID of the J2EE server region. For connections to other J2EE resources such as CICS and IMS, WebSphere for z/OS uses the RACF user ID mapped to the specified security role, regardless of the J2EE server property setting.

Running applications developed in WebSphere Studio Application Developer Integration Edition

The WebSphere Studio Application Developer Integration Edition tool can be used to develop applications (EJBs/servlets) which use connectors that are supported under WebSphere for z/OS and these applications can then be deployed and executed under a WebSphere for z/OS server. To support these applications, the WebSphere for z/OS server runtime includes the necessary jar files that are required to permit the execution of connector applications developed under WebSphere Studio Application Developer Integration Edition. Because of this, WebSphere Studio Application Developer Integration Edition developed applications which use connectors should not include any of the jar files that are required as a result of using the WebSphere Studio Application Developer Integration Edition tooling.

For more information on developing a J2EE application using one of the WebSphere for z/OS connectors, see WebSphere Studio Application Developer Integration Edition Help. See the section ″Samples-> IMS resource adapter″ for step by step instructions to create an application for WebSphere for z/OS connectors.

The jar files that are provided under the server in order to support WebSphere Studio Application Developer Integration Edition developed applications are as follows:

- Shipped in <*install_path*>*lib* where *install_path/lib* is the directory where you installed WebSphere Studio Application Developer Integration Edition.
  - jdom.jar
  - marshall.jar
  - physicalrep.jar
  - waswebc.jar
  - wsatlib.jar

  - wsif.jar
  - wsdl4j.jar
  - xerces.jar
- Shipped in <*install_path*> where *install_path* is the directory where you installed WebSphere Studio Application Developer Integration Edition.
  - soap.jar
  - xalan.jar

Notes:

1. When you configure a client that will invoke a connector application which was developed using WebSphere Studio Application Developer Integration Edition, you may need to include some or all of the above jars in the client's CLASSPATH.
2. Even though the wsif.jar is included to allow WebSphere Studio Application Developer Integration Edition developed connector applications to run under WebSphere for z/OS, composite service flows are not supported. Flow support is limited strictly to a simple flow to send a request to a single target resource manager.

Checklist for application components that use connectors

Table 12. Checklist for application components that use connectors

Check when completed:	**Task:**
Development tasks:	
☐	Use an appropriate development tool to create your J2EE application components. For example, use IBM's WebSphere Studio Application Developer Integration Edition or VisualAge for Java 4.0 to develop an Enterprise bean that uses connectors. For information about tools, see *Chapter 5. Setting up the application development environment*.
☐	Develop the application component. **Notes:** 1. For general information about developing components, see *Creating new application components to be installed in a J2EE server*. 2. If you are using WebSphere Studio Application Developer Integration Edition for application development, the tooling will handle the remaining development tasks listed in the following rows. Go to the "Assembly tasks" section of this checklist. 3. If you are using development tools other than Application Developer Integration Edition, you can find information about developing components to access CICS or IMS programs in the sample application instructions in the documentation for the WebSphere for z/OS-supplied CICSEXCI and IMSAPPC connectors. This documentation is available through download at theWebSphere Application Server Web page: `http://www.ibm.com/software/webservers/appserv/zos_os390/support.html`

Table 12. Checklist for application components that use connectors (continued)

Check when completed:	Task:
☐	Decide whether you want to implement access to a back-end resource by using a stateless session bean, or by integrating the function into the application business logic. **Recommendation:** For greater flexibility and reuse, isolate access to the back-end resource by using a stateless session bean.
☐	Choose a meaningful logical name to refer to the ConnectionFactory or DataSource you application will use.
☐	Use a JNDI lookup in `java:comp/env` for the ConnectionFactory or DataSource, using the logical name you chose as the reference name.
☐	When your application needs to use a connection factory, decide which type of connection is required for the back-end resource you want to access, and on the configuration used at your installation. **Rules:** In your application: • Use the appropriate ConnectionFactory reference types: – For the CICS Transaction Gateway ECI connector and IMS Connector for Java, use `javax.resource.cci.ConnectionFactory` – For DB2 JDBC and the IMS JDBC Connector, use `javax.sql.DataSource` • Use the appropriate interaction specs: – For the CICS Transaction Gateway ECI connector, use `com.ibm.connector2.cics.ECIInteractionSpec` – For the IMS Connector for Java, use `com.ibm.connector2.ims.ico.IMSInteractionSpec` For information about possible connector configurations, see *Overview of configuring the WebSphere for z/OS-supported connectors* in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834.
☐	Decide whether you want the application to provide the user ID and password to be associated with the connections that your application obtains. If so, indicate that the application is to be assembled with the `res-auth` deployment descriptor set to the appropriate value: • `Application` for Enterprise beans • `Servlet` for servlets For information about determining the user ID for resource authentication, see "Determining the user ID for resource authentication" on page 73.
☐	Decide whether you can safely cache a connection for later reuse, or whether you prefer to get, use, and then close the connection. **Recommendation:** Use the Get/Use/Close model for connection processing.
☐	Make sure you properly close connections so that you do not unnecessarily use up system resources. **Tip:** Keep in mind that resources like connections cannot be retained across end-of-life cycles for a bean (for example, across bean passivation and bean removal).
☐	Generate deployed code and package the components in the appropriate archive file (JAR or WAR files).
☐	Test the application components.

Assembly tasks:

Table 12. Checklist for application components that use connectors (continued)

Check when completed:	Task:
☐	Import tested components into a WebSphere application assembly tool."A WebSphere application assembly tool" means either the z/OS edition of the Application Assembly tool, the WebSphere Studio Application Developer tool, the WebSphere Studio Application Developer Integration Edition tool, or the Direct Deployment Tool/390fy
☐	Check the following deployment descriptor elements and extensions: • Transaction attribute • Connection management policy • RunAs identity and Set OS thread identity to RunAs identity • Resource authentication • ConnectionFactory reference types: – For the CICS Transaction Gateway ECI connector and IMS Connector for Java, use `javax.resource.cci.ConnectionFactory` – For DB2 JDBC and the IMS JDBC Connector, use `javax.sql.DataSource`
☐	Validate and deploy the application.
☐	Export the deployed application in an EAR file.
Configuration and installation tasks:	
☐	Configure the appropriate subsystems and connectors on z/OS or OS/390. For information about configuring subsystems and connectors, see *Configuring the WebSphere for z/OS-supported connectors* in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834.
☐	Define a new or modify an existing J2EE server, using the WebSphere for z/OS Administration application. **Tips:** • Under the following circumstances, make sure that connection management is configured into the sysplex: – If you plan to use J2EE connectors other than the DB2 JDBC datasource type – If you want to use connection management support, to better control DB2 JDBC connections and the consumption of DB2 resources • If you want to enable the J2EE server to reuse connections explicitly obtained in your application's code, make sure the JVM properties file for the server sets the `com.ibm.ws390.ConnectionUsageScopeDefault` property. For instructions for defining a J2EE server and connectors, see *Configuring the WebSphere for z/OS-supported connectors* in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834.

Table 12. Checklist for application components that use connectors (continued)

Check when completed:	Task:
☐	Add the connector to the J2EE server definition as a J2EE resource and resource instance, using the WebSphere for z/OS Administration application. **Rules:** • For the CICS Transaction Gateway ECI connector, use the `CICS_ECIConnectionFactory` type and appropriate J2EE resource instance properties. • For the IMS Connector for Java, use the `IMSConnectionFactory` type and appropriate J2EE resource instance properties. • For the IMS JDBC Connector, use the `IMSJDBCDataSource` type and appropriate J2EE resource instance properties. • To access DB2 through JDBC, use the `DB2datasource` type and appropriate J2EE resource instance properties. For instructions for defining J2EE resources, see the following topics in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834: • For the CICS and IMS connectors, see *Configuring the WebSphere for z/OS-supported connectors*. • For access to DB2 through JDBC, see the procedure for defining a J2EE resource in *Defining the BBOASR2 J2EE server*.
☐	Install the application, using the WebSphere for z/OS Administration application. During this process, associate the application component's resource references with the appropriate J2EE resource you defined for the connector.
☐	Validate and commit the J2EE server definition, and activate the J2EE server.

The WebSphere for z/OS environment for Web applications

Web components, which are known as Web applications, may consist of any combination of the following parts:
- One or more Java servlets
- Any other Java classes that act as utility classes in support of the servlets
- Static files such as HTML pages and GIF or JPEG images
- JavaServer Pages (JSPs) that format dynamic output

To enable Web applications for use, your Web-serving environment requires an HTTP handler, to receives HTTP requests from a network of browsers using the HTTP access protocol, and an execution environment, which interprets the inbound request and runs the appropriate servlet, based on the contents of the inbound request. The WebSphere for z/OS J2EE server includes a choice of two HTTP handlers and execution environments:
1. The HTTP and/or HTTPS Transport Handlers in combination with the Web container in the J2EE server, or
2. The IBM HTTP Server for z/OS in combination with the WebSphere for z/OS Local Redirector plug-in shipped with the WebSphere for z/OS product, and/or Web container in the J2EE server.

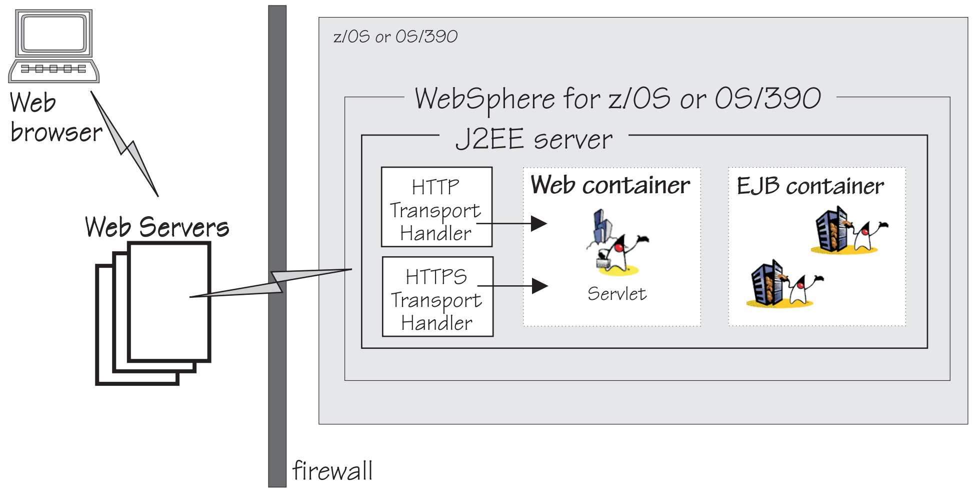


Figure 16. Possible configuration of the Web-serving environment on z/OS or OS/390

Web applications running in the Web container have direct access to resources on z/OS or OS/390, or can access them through Enterprise beans running in any WebSphere for z/OS J2EE server. Web applications use the RMI/IIOP protocol to access Enterprise beans running in J2EE servers on the same or different z/OS or OS/390 images.

Using the HTTP Transport Handler configuration

WebSphere for z/OS provides a J2EE server instance for J2EE application components: Enterprise beans and Web applications. A J2EE server should be configured to contain at least one EJB container and one Web container. The EJB container manages Enterprise beans, while the Web Container manages Web Applications.

The control region contains an IIOP Transport Handler that is able to receive IIOP requests and present them to the EJB container for processing. The IIOP Transport Handler is able to be configured via the System Management EUI.

The control region also contains an HTTP Transport Handler and an HTTPS Transport Handler that are able to receive HTTP(S) requests and present them to the Web container for processing. The HTTP and HTTPS Transport Handlers can be configured by placing BBOC_HTTP and BBOC_HTTPS environment variables in the WebSphere for z/OS current.env file.

HTTP client programs (such as browsers) can access Web Applications by sending requests to the HTTP or HTTPS Transport Handler. These Handlers route requests directly to the Web container for processing.

The configuration illustrated in Figure 16 uses the HTTP or HTTPS Transport Handler to handle inbound requests. The HTTP or HTTPS Transport Handler then passes the request to the J2EE server's Web container for processing. In this configuration, servlets run in the Web container, and can access Enterprise beans in the J2EE server's EJB container. This figure shows this connection between servlet and bean within the same J2EE server, but this connection can occur between different J2EE servers on the same or different z/OS or OS/390 images.

To correctly process a request:

- The appropriate properties must be set in the webcontainer.conf file when the Web container is defined within the J2EE server.
- The HTTP and/or HTTPS Transport Handler control region environment variables must be set in the current.env file during the WebSphere for z/OS customization process. See "Steps for enabling J2EE server support for Web applications (optional)" on page 147 for a description of these environment variables.
- The EAR file containing the requested application must be installed in the J2EE server.

These files are illustrated in Figure 17. The following list summarizes, in general terms, the contents of those required configuration files and how they relate to the processing of inbound requests. More details are provided through an example in "Resolving requests to a specific Web application" on page 84.

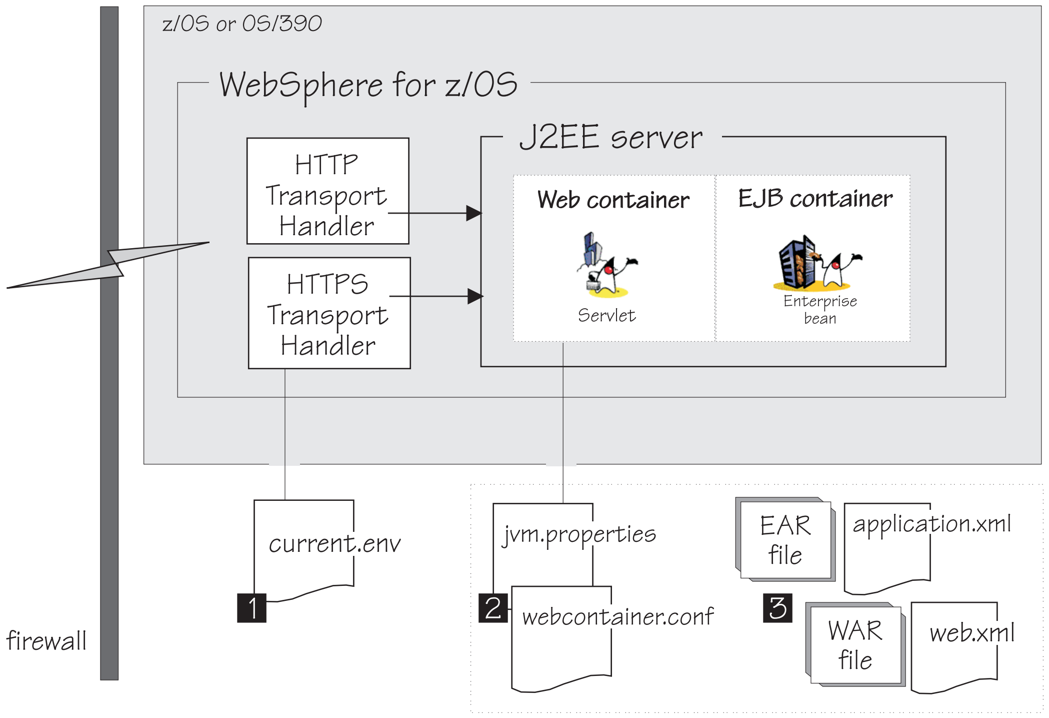


Figure 17. Files required when using the HTTP and/or HTTPS Transport Handlers

- The current.env file contains the environment variables defining the HTTP and/or HTTPS Transport Handlers. See "Steps for enabling J2EE server support for Web applications (optional)" on page 147 for a description of these variables and their default settings.
- The webcontainer.conf file contains definitions that enable the J2EE server's Web container to isolate Web applications, using constructs called virtual hosts and context roots:

  Virtual hosts
  Virtual hosts are named constructs, or aliases, that are equated with one or more internet hosts. These aliases enable the Web container to logically separate one or more Web applications from others installed in the same container. Each virtual host may be equated to multiple internet hosts.

  Note: If you are using the IBM HTTP Server for z/OS as your HTTP handler, you must make sure that the values specified for the

host.<*virtual-hostname*>**.alias** properties in the V4.0.1 was.conf file match values specified on the **host.default_host.alias** property in the webcontainer.conf file. In the V4.0.1 was.conf file, this property is initially set to **localhost**.

Context roots
Context roots are qualifiers in an inbound request (URL) that bind a Web application to a specific virtual host. This binding means that an inbound request for a servlet is executed only when the inbound URL contains a context root associated with the servlet and its virtual host.

The jvm.properties file, which is associated with a specific J2EE server instance, contains a pointer to the webcontainer.conf file.

- The .xml files related to the Web application. As described in Chapter 2, "Overview of application development and tools," on page 9, you need to package a Web application in a Web Archive (WAR) file, and then package this WAR file in an Enterprise Archive (EAR) file, perhaps along with the Enterprise beans (and their JAR files) that your Web application uses. Both WAR and EAR files contain Extensible Markup Language (XML) files that describe each component in the application. These XML files enable the WebSphere for z/OS J2EE server to provide the correct execution environment for the application components.

  The content of the XML files is something that you supply when you code and package the application, but the files themselves are generated for you.

Setting up the HTTP/HTTPS Transport Handler

If you are using the WebSphere for z/OS HTTP and/or HTTPS Transport Handler to receive inbound servlet requests for Web applications to be installed in this J2EE server, you must use the WebSphere for z/OS Administration application to include one or both of the following sets of environment variables in your current.env file. These environment variables are described in Appendix A, "Environment and JVM properties files," on page 299:

- The following BBOC_HTTP variables are required for the HTTP Transport Handler, which handles non-SSL requests:
  - `BBOC_HTTP_IDENTITY`
  - `BBOC_HTTP_INPUT_TIMEOUT`
  - `BBOC_HTTP_LISTEN_IP_ADDRESS`
  - `BBOC_HTTP_MAX_PERSIST_REQUESTS`
  - `BBOC_HTTP_OUTPUT_TIMEOUT`
  - `BBOC_HTTP_PERSISTENT_SESSION_TIMEOUT`
  - `BBOC_HTTP_PORT`
  - `BBOC_HTTP_TRANSACTION_CLASS`

  The following BBOC_HTTPS environment variables are required for the HTTPS Transport Handler, which handles SSL requests:
  - `BBOC_HTTP_SSL_IDENTITY`
  - `BBOC_HTTP_SSL_INPUT_TIMEOUT`
  - `BBOC_HTTP_SSL_LISTEN_IP_ADDRESS`
  - `BBOC_HTTP_SSL_MAX_PERSIST_REQUESTS`
  - `BBOC_HTTP_SSL_OUTPUT_TIMEOUT`
  - `BBOC_HTTP_SSL_PERSISTENT_SESSION_TIMEOUT`

  - BBOC_HTTP_SSL_PORT
  - BBOC_HTTP_SSL_TRANSACTION_CLASS

If the HTTPS Transport Handler is going to be used, configure your J2EE server for SSL support. See *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization* for a description of how to configure your J2EE server for SSL support.

Optionally, you can also include one or more of the following environment variables, depending on the needs of your installation. These environment variables are also described in Appendix A, "Environment and JVM properties files," on page 299:

- BBOC_HTTPALL_NETWORK_QOS and BBOC_HTTPALL_TCLASS_FILE, if you want to classify outbound data that is delivered in response to HTTP and HTTPS requests. Setting up these environment variables enables outbound data to be sent according to specified Quality of Service (QOS) TCP/IP packet prioritization.

  Note: This environment variable is only effective if you are running WebSphere for z/OS on z/OS Version 1 Release 2 or higher. It will be ignored for lower releases.
- BBOC_HTTP_SSL_CBIND=ON, if you want to require that all SSL connections, from a Web server with a Web server plug-in installed, include a specific client certificate in order for WebSphere for z/OS to accept that connection. WebSphere for z/OS will validate this client certificate against the RACF CBIND class.

  Setting up this requirement allows the establishment of a trusted and fully encrypted proxy relationship between the Web server and a WebSphere for z/OS J2EE server. This relationship makes it possible for sensitive information, such as the ″real″ client certificate contained in an AE Private Header, to be redirected from a browser to a WebSphere for z/OS J2EE server.
- BBOC_HTTP_MODE=INTERNAL or BBOC_HTTP_SSL_MODE=INTERNAL, if you want private headers received from a Web server plug-in, over the port specified on the BBOC_HTTP_PORT environment variable, to be trusted. See "WebSphere plug-ins for Web servers support" on page 92 for more information about this function.

Resolving requests to a specific Web application

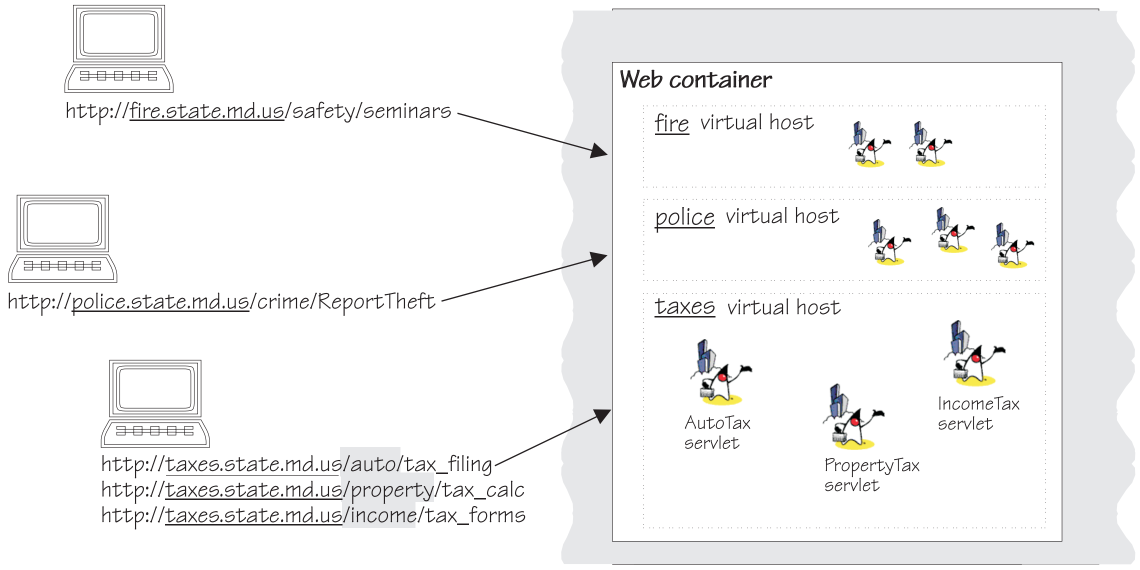
To understand how the HTTP(S) Transport Handler and the Web container work together, and use information in configuration files to satisfy a Web application request, consider the following example illustrated in Figure 18 on page 85. Suppose that your installation wants to do the following:

- Set up a Web site for the state of Maryland police, fire, and tax authorities.

  Each of the three authorities will have its own Web page and supporting Web applications. The left side of the figure depicts the three Maryland state Web pages: one for the fire authority, one for the police, and one for the tax authority. Under each is the URL that will appear on inbound servlet requests; the URL contains the domain name for each state authority page (the domain names are underlined in the figure). The URL also contains information that identifies the servlet to be run (this information is shaded in the figure).
- Host these different Web applications on the same z/OS or OS/390 system.

  Although these Web applications will be installed in the same Web container, your installation wants to keep the applications isolated, or logically grouped, by each state authority. The right side of the figure depicts the logical configuration that your installation wants to achieve. In this configuration, the Web container has three logical partitions that correlate with the domain names for each Web page: One for the fire authority, one for the police, and one for the tax authority. These partitions are virtual hosts, which separate each authority's Web applications from any other Web applications defined to this Web container. These virtual hosts also ensure that each authority's servlets are run only in response to requests that contain the appropriate domain name.

- Use the HTTPS Transport Handler to handle requests for these applications.



http://fire.state.md.us/safety/seminars
http://police.state.md.us/crime/ReportTheft
http://taxes.state.md.us/auto/tax_filing
http://taxes.state.md.us/property/tax_calc
http://taxes.state.md.us/income/tax_forms
Web container
fire virtual host
police virtual host
taxes virtual host
AutoTax servlet
PropertyTax servlet
IncomeTax servlet

Figure 18. Sample goals for using virtual hosts

The key to accomplishing these goals is correctly defining virtual hosts and context roots for each of the three authorities. Using the Maryland state tax authority Web page as a model, suppose that this tax authority page allows users to request information about income taxes, property taxes, or automobile taxes. In other words, inbound requests from the tax authority page, with domain name `taxes.state.md.us`, drive the IncomeTax servlet, the PropertyTax servlet, or the AutoTax servlet, as illustrated in Figure 18. In this case, your installation needs to include statements in the `webcontainer.conf` configuration file:

- Define a virtual host that correlates the `taxes.state.md.us` domain name on inbound requests, using a host alias property:
  ```
  host.taxes.alias=taxes.state.md.us
  ```
  This statement defines a virtual host named `taxes` in the Web container.
- Define the context roots for all Web applications that are associated with the tax authority domain. To do this, you use the host context-roots property:
  ```
  host.taxes.contextroots=/auto, /property, /income
  ```
  These context roots bind particular servlets to the `taxes` virtual host.

With these two property definitions in your Web container configuration file, you now have Web container constructs that match key elements of the URL in inbound servlet requests from the tax authority Web page:

- The virtual host `taxes` matches the underlined domain name `taxes.state.md.us` in the figure
- The context roots `/auto`, `/property`, and `/income` match the shaded information in the figure.

The virtual hosts and context roots definitions show you how the Web container handles inbound requests, but the HTTPS Transport Handler also needs to be configured so it knows what port to listen on for inbound requests to the Web container. The following environment variables need to be set in the current.env file:

- `BBOC_HTTP_SSL_LISTEN_IP_ADDRESS=nn.x.y.z.`
- `BBOC_HTTP_SSL_PORT=8090`

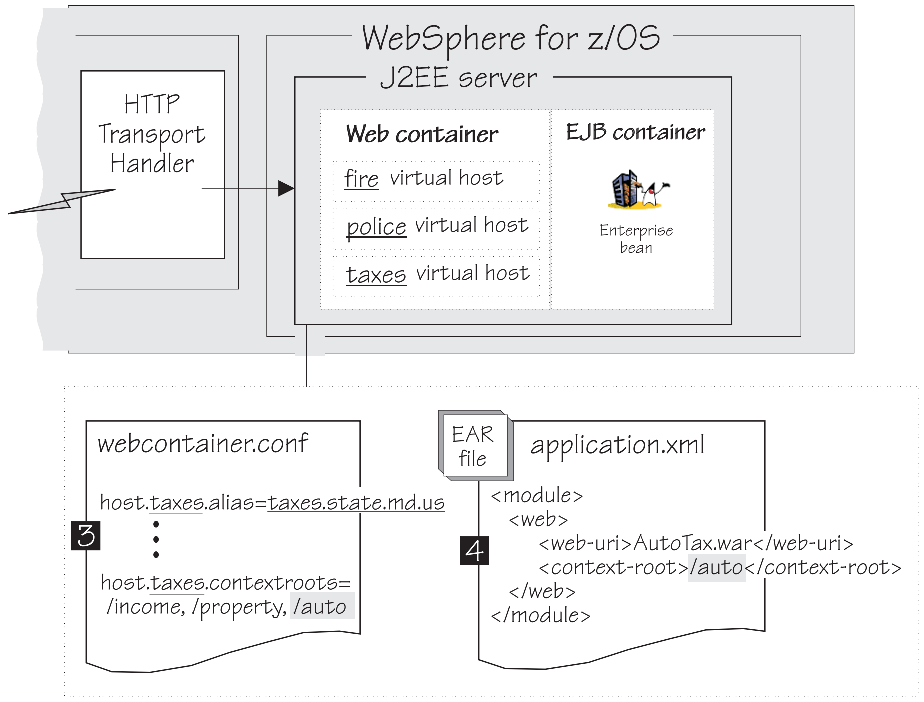


Figure 19. Resolving requests to a specific servlet in a Web application

Figure 19, and the following list illustrate the additional configuration information required to process an inbound request from the Maryland state tax page. Numbers in the list correspond to the numbers in the figure.

1. A user browses the Web site for the Maryland state tax authority, and submits a request for information about automobile taxes. The inbound request (URL) is:`http://taxes.state.md.us/auto/tax_filing`

   The Domain Name Server directs the request to the appropriate computer system: The Domain Name Server finds a DNS entry for `taxes.state.md.us`, and passes the request to IP address `nn.x.y.z.`, which is the adapter of the z/Series machine you want to use for Web-serving.
2. The HTTPS Transport Handler recognizes the inbound request and passes it to the Web container for execution.

   The HTTPS Transport Handler at IP address `nn.x.y.z.` listens at port 8090 and catches the inbound request. The HTTPS Transport Handler uses the following two properties in the `webcontainer.conf` file to match the domain name on the inbound request to a virtual host defined to the Web container:
   - The `host.taxes.alias` statement, which equates the name of a virtual host to the URL host name on an inbound request (`taxes.state.md.us`), and
   - The `host.taxes.contextroots` statement, which binds specific servlets to this virtual host.

At this point, the HTTPS Transport Handler knows the inbound request is valid for this execution environment, because the domain name and application context in the request match a defined virtual host and its associated context roots, respectively. The HTTPS Transport Handler passes the request to the Web container in the J2EE server.

3. The Web container in the J2EE server finds the actual Web application to run in response to the inbound request.

   The Web container uses the context root on the inbound request to find the servlet to run. It scans the XML files of applications installed in the J2EE server to find a matching context root value. The application XML file identifies individual modules in each installed Web application, including context root definitions for each module. The Web container finds the context root`/auto` in the XML for the servlet `AutoTax`; this context root matches the root specified on the inbound request. The Web container now has access to the `AutoTax` servlet's code, and can dispatch the servlet for execution.

HTTP session support

WebSphere for z/OS provides facilities under the heading Session Manager that support the javax.servlet.http.HttpSession interface described in the Servlet API specification. A session is a series of requests originating from the same user, at the same browser, that require access to a set of application-defined state data. WebSphere for z/OS:

- Fully supports the HTTP Session state semantic proposed by the Java Servlet Specification V2.2. It ensures that requests that are part of the same HTTP Session are not allowed concurrent access to session data in multiple server regions. If two requests that are part of the same session arrive at two different server regions, WebSphere for z/OS will serialize access to this session data across these server regions.
- Allows multiple requests for the same session data to execute concurrently within the same server region. It is the responsibility of the application components (servlets, JSPs, etc.) to serialize their access to the HTTP Session object within a server region. WebSphere for z/OS maintains the responsibility of providing the serialization among multiple server regions.
- Supports persisting HTTP session state data in a DB2 database. Depending on the needs of your installation, the WebSphere for z/OS administrator can use either DB2 Session Persistence Version 1 or Version 2 to manage how your installation's session data will be maintained in this database. (See "Using Persistent sessions" on page 88 for more information about these two versions.)
- Supports maintaining HTTP session state data in-memory. However, if the local redirector plug-in, provided with WebSphere for z/OS, is handling application requests, the session data can only be maintained within a single J2EE server instance environment, and only one server region can be defined for that sever instance. If an HTTP(S) Transport Handler is handling application requests, the session data can be maintained in a multiple J2EE server instances environment with multiple server regions defined for each server instance. (See "Maintaining session data In-memory" on page 89 for more information about maintaining session data in-memory.)
- Defines the notion of a session transaction. A session transaction begins when a servlet calls javax.servlet.http.HttpServletRequest.getSession(boolean). It ends with the completion of that servlet's javax.servlet.http.HttpServlet.service(request, response) method.
- Supports the use of a sessionshare.xml file to enable session data to be shared by Web modules within a J2EE application. This support is an IBM extension to the

Servlet 2.2 API Specification. Session data can not be shared across multiple J2EE applications. (See “Configuring session data sharing within a J2EE application” on page 170 for a description of this file.)

Note: In order to use this support, you must either be maintaining session data in memory or using DB2 Session Persistence Version 2 to manage your session data in a DB2 database.

The ability to constrain the number of run-time instances of a J2EE server is controlled by OS/390 Workload Manager policy.

For a description of how to configure HTTP Session State see “Steps for configuring HTTP Session Support” on page 160.

Using Persistent sessions

Persistent sessions use a DB2 database to maintain session data. WebSphere for z/OS provides two versions of session persistence:

- Version 1 uses a DB2 database as the mechanism for serializing access to and sharing HTTP session data. It uses the same DB2 database, tablespace, and table format as Versions 3.02, 3.5, 4.0 and 4.0.1 without Service Level 401401 installed. Therefore, the DB2 administrator is not required to create a new database for Version 4.0.1 if this version of DB2 session persistence is used. This version should only be used if you require applications running on Versions 4.0.1, 4.0, 3.5 and/or 3.02 of the WebSphere Application Server for z/OS and OS/390 to concurrently utilize the same database in their processing.
- Version 2 stores HTTP session data in an in-memory cache. It uses a DB2 database as the mechanism for:
  - Backing up the session data being maintained in this cache in case of a server region failure.
  - Preserving session data when the in-memory cache is full. (When the in-memory cache becomes full, the least recently used session data will be removed from the cache, leaving the copy of that data that is being maintained in the DB2 database as the copy that will be retrieved if it is needed.)

    It uses the same DB2 database format as WebSphere Application Server for Distributed Platforms V4.0 and higher, and provides improved performance and added functionality over Version 1. It requires that:
    - An HTTP(S) Transport Handler be used to handle J2EE application requests.
    - New DB2 session database, tablespace, and table be defined. (The name of the new table must be specified on the webcontainer.conf `session.dbtablename` property.)

    If you want to maintain session data across multiple J2EE server instances, you must also either:
    - Install a Version 5.3 IBM HTTP Server on a z/OS or OS/390 system and use FTP or another file transfer mechanism to download the WebSphere HTTP Plug-in for z/OS from your WebSphere for z/OS system to that HTTP Server, or
    - Install a WebSphere plug-in for Web servers on a supported non-z/OS Web server running on a distributed platform workstation.

    Then configure the Web server and the plug-in to communicate with the appropriate WebSphere for z/OS J2EE server instances.

You should be aware of the following caveats regarding how session management works within a multiple server region persistent session environment:

– During the processing of a session transaction, if the server region fails, the update to the DB2 database may not occur. 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. Once the transaction is completed, the session is still accessible regardless of the server region failure.
– If you add an object to a session that does not implement the serializable interface, there is no way to persist the object along with a given session. Consequently, the object will not be sent to and from the database when session updates are made. To make your applications portable to a multiple server region environment with persistent sessions, you must make objects placed in a session serializable.
– When HttpSessionBindingListener and HttpSessionBindingEvent are used in a multiple server region environment, the event will be fired in the server region where the session is currently being processed. This will occur when:
  - 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 all of the associated Session Managers

See "Steps for configuring HTTP Session Support" on page 160 for a description of how to configure DB2 session persistence.

Maintaining session data In-memory

If the local redirector plug-in is handling application requests, WebSphere for z/OS can only maintain session data for an environment consisting of a single J2EE server instance for which a single server region has been defined. In a multiple server region environment, the local redirector plug-in may not route requests for a specific HTTP session back to the server region that is maintaining the data for that session. It merely routes the request to the next available server region, which may or may not be the correct one.

If an HTTP(S) Transport Handler is handling application requests, WebSphere for z/OS can maintained session data for an environment consisting of multiple J2EE server instances, as well as multiple server regions within each server instance. Therefore, it is recommended that an HTTP(S) Transport Handler be used to handle requests for applications requiring session data to be store in-memory.

When an HTTP(S) Transport Handler is being used to handle application requests in a single J2EE server instance environment, WebSphere for z/OS can route requests for a specific HTTP session back to the server region maintaining the data for that session, even if multiple server regions have been defined for that server instance. In order for WebSphere for z/OS to maintain session data for an environment consisting of multiple J2EE server instances, you must do one of the following:

- Install a Version 5.3 IBM HTTP Server on a z/OS or OS/390 system and use FTP or another file transfer mechanism to download the WebSphere HTTP Plug-in for z/OS from your WebSphere for z/OS system to that HTTP Server, or
- Install a WebSphere plug-ins for Web servers (also referred to simply as a Web server plug-in) on a supported Web server running on a distributed platform workstation.

SeeTable 13 on page 93 for a list of supported non-z/OS Web servers and Web server plug-ins.

Once the Web server and plug-in are properly installed and configured, you can route requests from your browser, through the Web server and plug-in, to one of the WebSphere for z/OS J2EE server instances defined by a ServerGroup element in the plug-in's plugin-cfg.xml file. (See "Steps for configuring HTTP Session Support" on page 160 for a description of this file.) New requests will be randomly sprayed across the defined WebSphere for z/OS J2EE server instances.

Whenever a session is established, the WebSphere for z/OS Web container assigns a CloneID to that session, using either the default of <ServerName.ServerInstanceName> or the value set in the CLONEID property in the webcontainer.conf file. This CloneID will be used to ensure that future requests that occur during the same session will be routed back to the correct server instance, where the HTTP Transport Handler will route each request to the correct server region within that server instance.

Note: If your installation exploits HTTP Session Affinity (i.e., Session-in-memory using cookies over the HTTP(S) Transport Handler), you must disable the Server Region Recycle Function by either:

- Setting the Server Region Recycle Limit to 0 (zero) or
- Opening a new conversation for the J2EE server in the WebSphere for z/OS Administration application and uncheck ″Allow server region recycling″.

Use of the z/OS WLM operator command **V WLM,APPLENV=**<generic_server_name>**,REFRESH** is discouraged because existing server regions may **NEVER** terminate.

Session security

Maintaining the security of individual sessions is part of the function of the overall security structure built into WebSphere for z/OS. When creating a session as part of request processing, WebSphere for z/OS uses 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) WebSphere for z/OS uses a value of ″anonymous″ to denote the user. When processing a getSession() request on behalf of a Servlet, WebSphere for z/OS 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 WebSphere for z/OS. 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″.**
Unauthenticated HTTP request used to retrieve a session.	A new session is created and the user name is marked as ″anonymous″.	The session is returned.	The session is not returned; UnauthorizedSession RequestException is thrown.	The session is not returned; UnauthorizedSession RequestException is thrown.
HTTP request authenticated, 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; UnauthorizedSession RequestException is thrown.

Using cookies for session tracking

If cookies are enabled, the Session Manager will use a unique session ID to match user requests with their HttpSession objects on the server. When the user first makes a request and the HttpSession object is created, 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.

Using URL rewriting instead of cookies

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.

Note: If you have APAR PQ67436 installed on your system, URL rewriting can be used even if you are:

- Maintaining session data in memory across multiple server regions within a single J2EE server instance.
- Maintaining session data in memory across multiple J2EE server instances, provided you are using one of the following WebSphere plug-ins for Web servers:
  - The WebSphere HTTP Plug-in for z/OS that is used with the Version 5.3 IBM HTTP Server for z/OS and OS/390. (This plug-in is provided in WebSphere for z/OS Service Level W401500.) This plug-in (ihs390WASPlugin_http.so) should not be confused with the Local Redirector plug-in (was400plugin.so) that is shipped with the WebSphere for z/OS product to provide an IIOP connection from an IBM HTTP Server for z/OS and OS/390 to a WebSphere for z/OS Web container.
  - One of the WebSphere plug-ins for Web servers that run on a non-z/OS Web server that is shipped with WebSphere for z/OS and with the WebSphere Application Server Advanced Edition Version 4.0.5 or higher product. See Table 22 on page 176 for a list of these plug-ins.

The following example shows how Java code may be embedded within a JSP:

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

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 J2EE 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 for z/OS 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.

WebSphere plug-ins for Web servers support

WebSphere for z/OS includes plug-ins to allow connections via the HTTP(S) Transport Handler between a WebSphere for z/OS Web container and either:

- A Version 5.3 IBM HTTP Server for z/OS and OS/390, or
- A supported Web server running on a non-390 workstation platform. (Table 13 on page 93 lists the supported Web servers.)

These plug-ins are referred to as IBM WebSphere plug-ins for Web servers, or Web server plug-ins. The WebSphere HTTP Plug-in for z/OS is only available with the WebSphere for z/OS product. The plug-ins that are used with Web servers running on non-390 workstation platforms are provided with both WebSphere for z/OS and the AE version of the WebSphere Application Server for Distributed Platform product.

Note: If you use one of these plug-ins shipped with the WebSphere Application Server Advanced Edition Version 4.2 or higher, just follow the set-up instructions provided in "Steps for setting up WebSphere plug-ins for Web servers for use with WebSphere for z/OS" on page 171. You do not have to set-up the rest of the Advanced Edition product.

Once one of these plug-ins is installed on the appropriate Web server, servlet and JSP requests from that Web server can be redirected, via a WebSphere for z/OS HTTP(S) Transport Handler to WebSphere for z/OS where J2EE Web container functions are supported.

The WebSphere HTTP Plug-in for z/OS is located in the `/usr/lpp/WebSphere/WebServerPlugIn/bin` directory. "Setting up the WebSphere HTTP Plug-in for z/OS" on page 171 describes how to set up this plug-in to communicate with the HTTP(S) Transport Handler. This plug-in (ihs390WASPlugin_http.so) should not be confused with the previously existing Local Redirector plug-in (was400plugin.so) that is shipped with the WebSphere for z/OS product to provide an IIOP connection from an IBM HTTP Server for z/OS and OS/390 to a WebSphere for z/OS Web container.

Each of the non-z/OS Web server plug-ins that are supported in this configuration run on a number of operating system platforms. The plug-in files for each platform/Web server combination are stored in a separate WebSphere for z/OS subdirectory. The following table lists the non-z/OS Web server and platform configurations supported by these plug-ins, as well as the subdirectory where they are located within the WebSphere for z/OS product. "Setting up the Web server plug-in for a non-z/OS Web server" on page 175 describes how to set up these plug-ins to communicate with the HTTP(S) Transport Handler.

Table 13. WebSphere plug-ins for non-z/OS Web servers provided with WebSphere for z/OS

Operating system	Web server	Subdirectory
Windows 2000/NT	IBM HTTP Server (IHS)	/usr/lpp/WebSphere/Download Plugins/Win32/IHS
	Lotus Domino	/usr/lpp/WebSphere/Download Plugins/Win32/Domino
	Apache	/usr/lpp/WebSphere /Download Plugins/Win32/Apache
	iPlanet (Netscape)	/usr/lpp/WebSphere/DownloadPlugins/Win32/iPlanet
	Microsoft Internet Information Server (IIS)	/usr/lpp/WebSphere/DownloadPlugins/Win32/IIS
IBM AIX	IBM HTTP Server (IHS)	/usr/lpp/WebSphere/DownloadPlugins/AIX/IHS
	Lotus Domino	/usr/lpp/WebSphere/DownloadPlugins/AIX/Domino
	Apache	/usr/lpp/WebSphere/DownloadPlugins/AIX/Apache

Table 13. WebSphere plug-ins for non-z/OS Web servers provided with WebSphere for z/OS (continued)

	iPlanet (Netscape)	/usr/lpp/WebSphere/DownloadPlugins/AIX/iPlanet
HPUX	IBM HTTP Server (IHS)	/usr/lpp/WebSphere/DownloadPlugins/HPUX/IHS
	Lotus Domino	/usr/lpp/WebSphere/DownloadPlugins/HPUX/Domino
	Apache	/usr/lpp/WebSphere/DownloadPlugins/HPUX/Apache
	iPlanet (Netscape)	/usr/lpp/WebSphere/DownloadPlugins/HPUX/iPlanet
Sun Solaris	IBM HTTP Server (IHS)	/usr/lpp/WebSphere/DownloadPlugins/Solaris/IHS
	Lotus Domino	/usr/lpp/WebSphere/DownloadPlugins/Solaris/Domino
	Apache	/usr/lpp/WebSphere/DownloadPlugins/Solaris/Apache
	iPlanet (Netscape)	/usr/lpp/WebSphere/DownloadPlugins/Solaris/iPlanet
LINUX	IBM HTTP Server (IHS)	/usr/lpp/WebSphere/DownloadPlugins/LINUX/IHS
	Apache	/usr/lpp/WebSphere/DownloadPlugins/LINUX/Apache

Once you are set up to use a WebSphere plug-in for Web servers, in addition to regular plug-in functions, you can use private headers as a mechanism for forwarding proxy information from these plug-ins to WebSphere for z/OS, which would not otherwise be included with the HTTP requests. Private headers are implemented as a set of HTTP request header name/value pairs that the plug-ins add to the HTTP request header as it is forwarded by the Web server. The WebSphere for z/OS Web container removes this information from the header, and processes it. The private headers can include such information as the remote (client) user, the remote (client) host name, or an SSL client certificate. They conform to a naming standard so that there is no namespace collision with the architected HTTP header fields (hence the name ″private″).

For example, authentication information such as a client certificate, is normally requested by the Web server once during the establishment of an HTTP session, and then is not required again for individual requests within that session.

However, when these requests are forwarded to WebSphere for z/OS, the client certificate needs to be forwarded with each request, so that WebSphere for z/OS can use it as needed.

Similarly, the Web server examines the TCP/IP socket connection for information about the host address of the client. WebSphere for z/OS cannot do this because it’s socket connection is with the plug-in, not the actual client. Therefore, another one of the private headers is the host address of the actual client.

See “Steps for setting up WebSphere plug-ins for Web servers for use with WebSphere for z/OS” on page 171 for additional information.

Trust association interceptor support

WebSphere for z/OS is capable of performing both the authentication of Web clients and the validation that these requestors have been granted access to the appropriate role before allowing access to the requested URL. This processing, which is handled by the Web container in the J2EE server, is based on the security constraints specified by the deployment descriptor contained in the the target Web application’s web.xml file.

In addition to the authentication and authorization processing the Web container provides, your installation might want to use an external security product to perform authentication. WebSphere for z/OS enables the use of this type of external product through its trust association interceptor (TAI) support.

A trust association interceptor is Java code that can be configured for use by WebSphere for z/OS at run time. When WebSphere for z/OS determines that it needs to perform authentication processing, it sends the input request to a configured trust association interceptor. The interceptor examines the content of the request and returns a string, containing the name of a user within the configured user registry. WebSphere for z/OS then treats the user as authenticated and makes that user name the principal of the current request. Any necessary access checks will be performed based on the permissions that have been granted to the authenticated user in this environment. If a trust interceptor does not indicate it has authenticated a user, WebSphere for z/OS will perform authentication according to the rules specified by the deployment descriptor in the web.xml file for the requested application.

Your installation might want to use a trust association interceptor if it has a third party security product acting as a reverse proxy in a DMZ. This third party product performs authentication of the Web clients within the DMZ and then forwards the request to WebSphere for z/OS for processing. The trust association interceptor that the third party security product provides must implement the TrustAssociationInterceptor class required by WebSphere for z/OS. This class, which is located in the Java package com.ibm.websphere.security, enables the third party product to indicate to WebSphere for z/OS that authentication processing has already been performed and to identify the authenticated user to WebSphere for z/OS. This prevents WebSphere for z/OS from redundantly trying to authenticate the client.

WebSphere for z/OS does not know how the interceptor authenticates a user. It also doesn’t know how the trust association interceptor determines that the request has been forwarded by a trusted proxy server. The method by which the trust association interceptor determines that the request has been forwarded by a trusted proxy server is determined by the trust association interceptor provider.

Trust association interceptors are defined to the WebSphere for z/OS run-time environment using properties in the webcontainer.conf file. Multiple versions of these properties can be included in the webcontainer.conf file if you need to define multiple interceptors to the WebSphere for z/OS run-time environment. This capability makes it possible to set up configurations with multiple reverse proxy vendors that can all forward requests to the same WebSphere for z/OS instance for processing. When multiple interceptors are configured, WebSphere for z/OS calls them in the order in which they are defined. Once an interceptor indicates that it has authenticated a client, WebSphere for z/OS will not call any subsequent interceptors.

The implementation class for the trust association interceptor must be be defined to WebSphere for z/OS before it can be utilized at run time. Consult with the provider of your trust association interceptor for any special instructions for setting up or configuring their interceptor within a WebSphere for Z/OS run-time environment. (See the TrustAssociationInterceptor class contained in the Java package com.ibm.websphere.security for information on how to implement a Trust Association Interceptor.

Using a custom user registry with WebSphere for z/OS

WebSphere for z/OS provides a built-in authentication and authorization mechanism for Web clients. This mechanism, WAS390WebAuthMechanism, provides support for:

- Challenging Web clients for inputs according to the rules described by the J2EE v1.2 and the Servlet v2.2 specifications.
- Enabling single sign-on support to be provided among WebSphere v4.0.1 for z/OS or higher Application Servers, which are configured to use the same user registry and SSO values.
- Delegating authentication and authorization support to a third party authentication and authorization server that is able to provide a Trust Association Interceptor for use at runtime.

WebSphere for z/OS J2EE servers that are to service requests from Web clients can be configured as follows:

1. SAF based configuration

   This is the default configuration in which WebSphere authenticates users and groups to a SAF (RACF) security system. J2EE permissions are described via SAF EJBROLE resource profiles within the SAF system. Optimized access to z/OS Resource Managers, (including container managed client level access authority, is available with this configuration.
2. Custom user registry configuration

   This configuration option allows a third party user registry to be provided for use with WebSphere for z/OS. In this configuration, J2EE permissions are not configured within the SAF system. Instead they are configured using an authorization table. This table is contained in an XML file that is pointed to by the following set of tags contained in the XML file specified on the `WebAuth.CustomRegistry.authorizationTableXML=`*filename*`.xml` property in the webcontainer.conf file:

   ```
   <Application>
         <name>application_name</name>
         <permission-file-name>fully_qualified_filename
         </permission-file-name>
   </Application>
   ```

Note: There should be a set of these tags for each application installed on the J2EE server for which the custom user registry is being used.

When using a custom user registry in a WebSphere for z/OS environment, you should be aware of the following:

a. Authenticating remote EJB clients using a custom user registry is not supported. However, EJBs that are accessed from a Web application that is deployed in the same J2EE server as these EJBs can be administered within the domain of a custom user registry.
b. It is recommended that EJBs not be exposed to remote clients from a J2EE server which is configured to make use of a non-SAF registry.
c. When using single sign-on capability for an application, it is the responsibility of the administrator to ensure that all WebSphere for z/OS J2EE servers that are part of the sign-on domain are using the same registry (i.e., the same SAF User Registry or the same custom user registry).
d. Identities associated as a result of the EJB methods runAs server and runAs RoleName will not support custom user registry identities. Instead, they will use a SAF identity and subsequent authorizations will be done using the SAF EJBROLE profile.

The following table summarizes how permissions are obtained using SAF and using a custom user registry:

Table 14. Permissions summary

SAF	Custom user registry
Users and groups defined in RACF.	Users and groups are defined in a custom registry.
Roles are set up by defining profiles in RACF	Users and groups are defined in an authorization table.
Users and groups are permitted to roles by having access to the RACF profile. Use the RACF PERMIT command to grant additional roles permission to the EJBROLE profile.	Users and groups are permitted to roles through the authorization table. Edit the XML file containing the authorization table to authorize additional roles.

Methods used to define user registries

The following table summarizes the methods used to define user registries and the resulting mechanism for defining permissions.

Table 15. Summary of the methods used to define user registries and the resulting mechanism for defining permissions

Web clients	User registry	J2EE permissions	Resource adapter
WAS390Web AuthMechanism is used as a mechanism for client authentication and authorization, and for managing Web clients logins to WebSphere for z/OS J2EE servers. When a protected URL (as defined by security constraint in web.xml deployment descriptor) is to be accessed by a web client, Web Container will validate that the requestor has the required permissions (i.e. access to role). If necessary, clients are challenged for authentication data via the policy in web.xml deployment descriptor. The following challenge types are available: • User ID/password via HTTP Basic authentication • User ID/password via Form-Based Login. In addition, the Web container will consult with any configured Trust Association Interceptors prior to performing a challenge. Trust Association Interceptors can inform the Web container if a client is to be considered a current, authenticated principal. The Web container will also allow for any authenticated client to be delegated to the WebSphere for z/OS J2EE server by a Trusted Proxy Server such as Tivoli WebSeal.	SAF • When user ID/password is acquired via a Basic authentication or Form-Based login, it is validated against SAF. • When a client certificate is provided, the certificate is passed to SAF and resolved via Certificate Mapping Filters that are configured in the SAF system. The resultant SAF principal from the certificate mapping operation is considered the current principal. • When a user ID string is returned from a Trust Association Interceptor, the client is considered to be a valid user within the SAF system.	When using SAF as the user registry, J2EE permissions are defined via SAF EJBROLE resource profiles within the SAF system that contains the user registry. Users and groups are allowed permission to the J2EE server's resources when the J2EE server administrator grants the users/groups access to the resource profiles representing the ROLE. This is accomplished using SAF facilities.	SAF based resource adapters use the value specified on on a resource's deployment descriptor `res-auth` attribute to determine how authorization is to be handled: • For resources that were deployed with a deployment descriptor containing the attribute `res-auth=application` an application must include a user ID and password whenever it invokes the getConnection API. • For resources that were deployed with a deployment descriptor containing the attribute `res-auth=CONTAINER`, the Web container, in conjunction with the resource adapter, will establish the connection with a primary authorization ID equal to the ID of the SAF user that is the current principal. This attribute is configured on a per J2EE server basis.

Table 15. Summary of the methods used to define user registries and the resulting mechanism for defining permissions (continued)

	Third parties or installations may use a custom user registry by providing an implementation of the com.ibm.websphere.security. CustomRegistry interface. • When a user ID/password is acquired via Basic authentication or Form-Based login, it is validated against the custom user registry. The name returned via the checkPassword method of the custom user registry implementation is the name assigned to the current principal. • When a user ID string is returned from a Trust Association Interceptor, the client is considered to be a valid user within the custom user registry. This will be validated via a call by the runtime to the isValidUser() method.	When a custom user registry has been configured for use with WebSphere for z/OS, J2EE permissions are required to be defined to the J2EE server in an XML file containing an authorization table. This XML file allows the J2EE administrator to grant permission to J2EE roles on the basis of a user, a group, and the application which is being accessed.	The `res-auth` deployment descriptor attribute specified when a resource is deployed determines how authorization is to be handled: • Resources that are deployed with a deployment descriptor containing the attribute `res-auth=application` require an application to provide a user ID and password on input of getConnection API. • Resources that are deployed with a deployment descriptor containing the attribute `res-auth=CONTAINER` require the Web container, in conjunction with the resource adapter, to establish the connection using the SAF ID specified on the `WebAuth.CustomRegistry. SAFPrincipal` property in the webcontainer.conf file. The `res-auth=CONTAINER` attribute is configured on a per J2EE server basis.

Defining J2EE Permissions for J2EE servers that are configured for a non-SAF based user registry

When a WebSphere for z/OS J2EE server has been configured to make use of a custom user registry for authentication and authorization processing, the permissions that are granted to users and groups within that J2EE server are provided by the administrator via an XML file that contains an authorization table. This file describes the authorization rules that WebSphere for z/OS is to use at runtime. You can not use EJBROLE profiles defined in a SAF system if users and groups are defined in a non-SAF based registry.

The syntax of the XML file containing the authorization table allows the administrator to grant users and groups access to J2EE resources on a per application basis. It also allows the administrator to define a global set of permissions that are used for applications that do not have an application specific definition. At runtime, WebSphere for z/OS determines the roles that are required to access a J2EE resource. If the current user, or a group in which that user is a member, has been granted access to one of the required roles, then that user is permitted access to the J2EE resource.

Users and groups are permitted access by mapping them to the appropriate roles defined in an application's deployment descriptors. <role roleName> tag sets within the authorization table XML file are used to define these mappings. (See "Creating XML files containing authorization tables" on page 196.) Two special values that can be specified within this tag set are:

- `<user userName="AllAuthenticatedUsers"/>`, if you want to associate all authenticated clients with a role. When this value is specified, any successfully authenticated client will be mapped to that role.
- `<user userName="Everyone"/>` if you want to associate every client with a role. Clients do not have to be authenticated in order to get mapped to this role. When this value is specified, WebSphere for z/OS acts as though any application associated with this role is unprotected.

When determining a match for authorization rules, WebSphere for z/OS will first look for an exact match on a specific application name. If no application specific definition is provided, WebSphere for z/OS will look for a global permission definition defined by the wildcard "*" application name. There is no support provided for partial matches on application names.

If a matching application is not found, via an explicit match or a wildcard match, or if the role has not been provided within the matching application stanza, WebSphere for z/OS will assume that the permission has not been granted and will deny access to the resource.

The XML file containing the list of XML files containing authorization tables is defined to the WebSphere for z/OS J2EE server using a webcontainer.conf file property. Each server region within a J2EE server instance reads this file as part of its initialization. The J2EE server administrator is responsible for controlling access to this file. It is recommended that this file be allowed to be written and update by an administrator, and able to be accessed in read-only mode by the server region process at runtime. This is similar to existing recommendations for the was.conf file and the httpd.conf file.

Implications of accessing z/OS Resource Managers

WebSphere for z/OS provides support for optimized access to existing z/OS Resource Managers and subsystems such as CICS, IMS, and DB2/ESA. This support is provided in the form of configurable resource adapters that applications access through Standard J2EE programming APIs.

J2EE specifications allow the security policy, that is used when connecting to a resource manager, to be configurable. The security policy is determined by the setting specified on the `res-auth` attribute of a deployment descriptor for a Web application or EJB that needs to access the requested resource.

- When `res-auth=APPLICATION` is specified, the Web container and the EJB container expect the application include a user ID/password in the connection API. The resource adapter uses this user ID/password to verify that the application has permission to access the requested resource.
- When `res-auth=CONTAINER` is specified, the Web container and the EJB container, working in conjunction with the z/OS Resource Managers, establish a SAF principal as the primary authorization ID. The resource adapter uses this ID to obtain a connection to the requested resource. By default, when running in a J2EE server that is configured to make use of a custom user registry, the containers will work with the Resource Manager to establish the connection using the identity of server region hosting the application. This ID is also used when accessing EJBs that are not part of the same enterprise application but are deployed in the same J2EE server.

If you don't want to use the server region ID, you can update the `WebAuth.CustomRegistry.SAFPrincipal` property in the webcontainer.conf file to specify a valid MVS user ID that you want to used for establishing resource connections.

Using the CustomRegistry interface

The CustomRegistry interface, which is described in "Implementing the CustomRegistry interface" on page 199, defines a set of very general methods that are called to perform security operations for applications configured to use a custom user registry. When implementing the methods in the interface, an application developer must decide how to map the information manipulated by the CustomRegistry interface to the information in the custom user registry. The methods in the CustomRegistry interface operate on the following user information:

User name
: This is the name by which a user is authenticated using a checkpw call. The string returned from this call is used as the user name for subsequent authorization checks. The CustomRegistry interface requires user names to be unique. For most registries, the user name logically maps to an identifier that is meaningful to the user; some common terms for this identifier include login name, account name, user name, and principal.

Unique identifier
: The CustomRegistry interface requires this identifier to be unique. For most registries, the unique identifier logically maps to a numeric counterpart of a user name. WebSphere for z/OS automatically assigns a user ID (UID) to each user name.

The CustomRegistry interface also operates on parallel information for groups

Group name
: An identifier for a group. As with the user name, the CustomRegistry interface requires group names to be unique. For most registries, user names logically map to one or more groups. This enables either a user name or a group name to be used to authorize access to one or more roles.

Unique identifier
: A unique identifier for a group.

See "Steps for enabling a custom user registry" on page 193 for a description of how to enable a custom user registry within a J2EE server environment.

Batch compiling JSPs

As an IBM enhancement to JSP support, WebSphere for z/OS provides a batch JSP compiler tool called the JspBatchCompiler tool. This tool, which you initiate from an OMVS command line prompt, should be run when the J2EE server instance, specified on the server.name parameter is offline. Running this script while the server instance is running will impact application response time.

Batch compiling JSP files makes the J2EE server instance's response to the first request for a JSP much faster because the JSP has already been translated and compiled into a servlet before any request is received. Batch compiling is also useful as a fast way to resynchronize all of the JSP files for an application.

Dynamic fragment caching

A WebSphere for z/OS performance enhancement is the ability to cache the output of dynamic servlets and JSP files within a single server region. Working within an application server's Java Virtual Machine (JVM), this technology intercepts calls to a servlet's service method, and checks whether the invocation can be served from a cache. Because J2EE applications have such high read-write ratios and can tolerate a small degree of latency in the freshness of their data, fragment caching creates an opportunity for significant gains in server response time, throughput, and scalability, thus improving overall performance.

After a servlet is invoked once (generating the output that will be cached), a cache entry is created containing not only the output, but also side effects of the invocation, such as calls to other servlets or JSP files, as well as meta data about the entry including timeout and entry priority information. Unique entries are distinguished by an ID string generated from the HttpServletRequest object for each invocation of the servlet. Servlet caching can then be based on:

- Request parameters and attributes
- The URI used to invoke the servlet
- Session information
- Other options, including cookies

Since WebSphere for z/OS compiles JSP files into servlets, the dynamic cache function treats them the same.

Each server region has its own cache. Therefore, when a server region is recycled, the cache is cleared.

Note: If you are porting Web applications from a WebSphere Application Server for Distributed Platforms system that use dynamic fragment caching, you should be aware that:

- There is one restriction: the Servlet Cache Monitor application can only be used in a a J2EE server instance environment with a single server region defined.

  WebSphere for z/OS provides the Servlet Cache Monitor application as a tool for verifying that your servlets and JSPs are being properly cached. This tool enables you to inspect the contents and behavior of the fragment cache. However, if this tool is used in an environment that has more more than one server region configured (using MIN_SRS=nn) for a J2EE server instance, it may provide invalid results.
- The dynacache.xml file can not be configured using the WebSphere for z/OS Administration application.
- The same dynacache.xml and servletcache.xml files can be used with both the WebSphere Application Server for Distributed Platforms and the WebSphere for z/OS products.

External caching

WebSphere for z/OS's dynamic cache has the ability to control external caches on Web servers that are being used to perform external caching of servlets and JSP files.

When external caching is enabled, the cache matches pages with their URIs and pushes matching pages to the external cache. The entries can then be served from the external cache instead of the application server. This creates a significant savings in performance.

Only certain fragments are eligible for external caching. Since the external cache must use the full URI as a cache id, the servlet being externally cached uses that URI as its internal cache id as well. Also, because the the cache automatically uses the URI to build cache ids, it is illegal to define cache variables (for example, request, session, and cookie variables) in an externally cacheable servlet.

Only full pages are pushed out to external caches, so only externally accessible servlets should be defined as externally cacheable. For example, if page1.jsp includes page2.jsp and page3.jsp, then only page1.jsp should be declared externally cacheable. If page3.jsp is invalidated, then the cache also invalidates the external entry for page1.jsp. Therefore the next request for page1.jsp is sent to WebSphere Application Server.

Servlet and JSP file content that is private, requires authentication, or uses SSL should not be cached externally. The authentication required for those servlet or JSP file fragments cannot be performed on the edge. A suitable timeout value should be specified if the content is likely to become stale.

At this time, there are no products that implement an external cache adapter for the dynamic fragment caching support for WebSphere for z/OS.

Monitoring dynamic fragment caching

WebSphere for z/OS provides the Servlet Cache Monitor application for inspecting the contents and behavior of the fragment cache. (“Steps for enabling dynamic fragment caching” on page 204 describes how to set up this application.). Once you have the monitor running, you can view the statistics page which describes the following properties:

- Cache Size - The maximum number of cache entries, as defined in the dynacache.xml file.
- Used Entries - The number of entries currently contained in the cache. The cache is considered full even if this number is less than the number specified for Cache Size because the cache tries to keep 20% of the entries in reserve.
- Cache Hits - The number of servlet/JSP requests, both external (from a browser) and internal (included/forwarded from another servlet/JSP) that have been served from the cache since startup.
- Cache Misses - The number of requests for cacheable servlets or JSPs, that were not served from the cache, i.e. resulted in executing the fragment and storing the result in the cache.
- LRU Evictions - The number of entries that have been removed from the cache by the LRU algorithm when the cache was full and new entries needed to be added.
- Explicit Removals - The number of entries removed from the cache through cache policy rules, or through the cache monitor.
- Default Priority - The default value for a cache entry’s priority as defined in the dynacache.xml file. (See “Custom ID and MetaData generators” on page 104 for more information about priority.)

You can click on Contents to view the Current Cache Contents page and see a list of cache entries. For each cache entry listed you can:

- Click on an entry listed under the Template keyword to display a list of cache entries that have the same template. From this view, you can:
  1. Invalidate the entries for that template.
  2. Click on the dataIds field for the entry to display a list of cache entries that have the same dataId. From the dataIds view, you can invalidate the entries

with that dataId or click on an entry listed under the cacheEntry ID keyword to display a detailed description of that entry.
3. Click on the cacheEntry ID to display a detailed description of that entry.

- Click on an entry listed under the Cache ID keyword to display a detailed description of that entry.
- Click on an entry listed under the Data IDs keyword to display a list of cache entries that have the same dataId. From this view, you can invalidate the entries with that dataId or click on an entry listed under the cacheEntry ID keyword to display a detailed description of that entry.

From the detailed description view, you can invalidate the entry, reset its position in the LRU algorithm (mimicking a cache hit on that entry), or return to the main list of entries.

You can click on Clear Cache from any view to remove every entry currently in the cache.

Custom ID and MetaData generators

WebSphere for z/OS's dynamic cache technology allows users to replace the standard configuration functions with their own custom configuration classes. The configuration duties managed by the cache fall into two categories:

- Generating cache and group IDs
- Defining meta data such as timeout, priority, and external caching

Application developers can supply classes to handle either or both of these sets of responsibilities, by implementing

```
com.ibm.websphere.servlet.cache.IdGenerator
```

and

```
com.ibm.websphere.servlet.cache.MetaDataGenerator
```

Overriding the default MetaDataGenerator allows users to access configuration information from some other source, or makes timeout, priority, or external cache group a function of a variable rather than a constant. A new IdGenerator gives users the ability to determine cache entry ids and their group ids. Both classes can still use the cache policy attributes defined for a servlet (<timeout>, <priority>, <request>, and others) to relay data to their generators using the com.ibm.websphere.servlet.cache.CacheConfig class.

Each servlet class has individual IdGenerator and MetaDataGenerator objects associated with it. So if the same servlet is being executed by WebSphere Application Server in different threads, all threads will use the same pair of generator objects. Several dynamic caching classes are not described in detail here. For a full description of the com.ibm.websphere.servlet.cache API package, including the classes and interfaces used by the cache function, see the description of this API package at URL:

```
http://www.ibm.com/software/webservers/appserv/doc/v40/aee/wasa_common/apidocs/index.html
```

To configure the cache to use a custom ID generator, include a <idgenerator> tag in the servletcache.xml file to specify the IdGenerator.

```
<servlet>
  <timeout seconds="0"/>
  <path uri="/servlet/CommandProcessor" />
  <idgenerator class="SampleIdGeneratorImpl" />
</servlet>
```

| See "Building a custom ID generator" on page 216 for additional information.

Web services

WebSphere for z/OS uses SOAP (Simple Object Access Protocol) as its framework for supporting the creation and deployment of Web Services. Its SOAP implementation includes V2.2 of the Apache SOAP implementation, which is a Java-based implementation of the SOAP 1.1 specification. This implementation:

- Gives applications a simple way to take advantage of an increasing number of information sources that are becoming available over the Web by enabling them to open up an HTTP connection, invoke a remote method, and receive the resulting response.
- Enables you to expose EJBs as SOAP Services.
- Includes an implementation of the Security extensions for SOAP, which enables secure connections (see "Securing SOAP Services" on page 255).

This implementation of SOAP uses a set of XML tags to indicate the roles of different pieces of information being sent over the Web using the HTTP transport protocol. With SOAP, typically:

1. Your program passes the name of the remote method it wants to invoke, along with any required parameters, to your client library.
2. The library assembles the appropriate XML document, called a SOAP request, which includes a description of the method to be invoked and the required parameters.
3. The library passes this XML document to the SOAP server identified by a SOAP endpoint URL. (A SOAP endpoint URL is used, much the same way as when you point a browser at a Web server address by specifying the server's URL.)
4. The SOAP server attempts to execute the method.
5. It then assembles a SOAP response XML document around the result of the execution and passes it back to the SOAP client. (The response is either the result of the execution or the appropriate error message.)
6. Upon receiving the SOAP response, the client library parses the XML document to get the result of the method invocation and passes this result to the program using the library.

See Chapter 15, "Creating and deploying Web Services," on page 249 for a description of how to set up your Web services.

If you require a SOAP client, there are SOAP clients for most programming languages available for downloading from the internet.

Considerations for test and production environments

Before you put an application into production on z/OS or OS/390, it is important that you run the code in a WebSphere for z/OS server for testing. You may bring up the application and simulate a real load on the application. To do this, you need to define an additional **test server** and install the application into it. It is possible to have different levels of the application installed within the same sysplex inside different test and production servers.

Because WebSphere for z/OS is designed and built to take advantage of the sysplex environment, which includes such shared facilities as datasharing, intelligent workload balancing, and shared transaction management, there are opportunities and challenges for software testing due to this shared resource design. When choosing a test and production configuration, weigh the risks involved according to the resources shared between test and production systems. *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834, explains possible test and production system configurations and the risks involved for each.

Part 2. Creating, assembling and deploying J2EE server applications

Chapter 5. Setting up the application development environment

All of the tools you use to develop, assemble, and install J2EE application components are workstation tools. Because documentation for installing and using these tools is available already, the following procedure provides only a summary of the installation steps, with references to resources with further instructions. Use this procedure as a checklist to make sure you have the correct tools and information sources on hand, before you begin to develop J2EE application components.

Steps for setting up your workstation

Perform the following steps to set up your workstation for developing, assembling, and installing Java 2 Enterprise Edition (J2EE) applications:

1. Decide which application development tools and other software you will need to develop your application. Use the following table to help you determine which software is necessary.

© Copyright IBM Corp. 2000, 2003

Table 16. Software requirements for Java 2 Enterprise Edition application components

J2EE application component	Software to use
Enterprise beans	**For development:** One of the following: • WebSphere Studio Application Developer and 390fy, which is the preferred method of deploying applications. • The WebSphere for z/OS Application Assembly tool (read the recommendations following this table for further information about using the Application Assembly tool). • The IBM WebSphere Studio Application Developer V4.0x • WebSphere Studio Application Developer Integration Edition • Non-IBM tools, such as JBuilder or Visual Cafe, for application development. Use the documentation for those products to determine hardware and software requirements.
	For testing: One of the following: • IBM WebSphere Studio Application Developer V4.0x • WebSphere Studio Application Developer Integration Edition environment • This combination of products: – IBM or Sun Microsystems Java 2 Standard Edition (J2SE) Software Development Kit (SDK) V1.3 – WebSphere Application Server Advanced Edition, V3.5, or Advanced Single Server Edition, V4.0 (Optional) DB2 Universal Database Version 7.1, required only for testing beans that require the use of a persistent datastore.
	For assembly: One of the following: • The WebSphere for z/OS Application Assembly tool (read the recommendations following this table for further information about using the Application Assembly tool) • The IBM WebSphere Studio Application Developer V4.0x
	For installation in a J2EE server: • The WebSphere for z/OS Administration application
Servlets and JavaServer Pages (JSPs)	**For development and testing:** One of the following: • WebSphere Studio Application Developer and 390fy, which is the preferred method of deploying applications. • The WebSphere for z/OS Application Assembly tool (read the recommendations following this table for further information about using the Application Assembly tool). • The IBM WebSphere Studio Application Developer V4.0x • IBM or Sun Microsystems Java 2 Standard Edition (J2SE) Software Development Kit (SDK) V1.3
	For assembly: One of the following: • The WebSphere for z/OS Application Assembly tool (read the recommendations following this table for further information about using the Application Assembly tool) • The IBM WebSphere Studio Application Developer V4.0x
	For installation in a J2EE server: The WebSphere for z/OS Administration application

Recommendations:

- The preferred method of deploying applications is to use WebSphere Studio Application Developer and 390fy. If you are using the following two IBM extensions, however, you still need to use the Application Assembly tool:
  - **SyncToOSThread:** See "Synchronizing operating system thread identity to RunAs identity" on page 31 for more information on this extension.
  - **Connection Management Policy:** See "Exploiting connection management support" on page 71 for more information on the Connection Management Policy extension.
- Use the IBM WebSphere Studio Application Developer to develop and test beans, servlets, and JSPs. This product enables developers to fully test entity and session beans, including JNDI lookups, remote method calls, and method calls on the home interface. It also has a servlet engine, so that servlets and JSPs can be served up to a Web browser as if they were going through an HTTP and Application Server.

  Additionally, WebSphere Studio Application Developer enables you to automatically package servlets or JSPs into Web application archive (WAR) files. (If you use other tools, you might have to create the WAR files manually.)
- If you are using the WebSphere for z/OS Application Assembly tool, download the latest copy from the WebSphere Application Server web site (go to

  ```
  http://www.ibm.com/software/webservers/appserv/
  zos_os390/support.html
  ```

  and click `Download` on the left frame).

---

2. If necessary, install or upgrade the appropriate application development software on your workstation. For installation or migration instructions, see the following references:

Table 17. References for installation or migration information for application development software

Software:	**Source of installation or migration information:**

• IBM WebSphere Studio Application Developer
• WebSphere Advanced Edition for distributed platforms
	For hardware requirements and installation instructions, for IBM WebSphere Studio Application Developer, use a browser to view the Web page at:
	`http://www.ibm.com/software/ad/studioappdev/`
	
	For hardware requirements and installation instructions, for WebSphere Advanced Edition for distributed platforms, use a browser to view the Web page at:
	`http://www.ibm.com/software/webservers/appserv/library.html`
	
	From this web page, click on the following links:
	• Supported hardware, software, and APIs for WebSphere V3.5 or V4.0 (all editions for distributed platforms), for a complete listing of hardware and software.
	• The **InfoCenter** for WebSphere V4.0 for distributed platforms, for information about or links to planning and installation instructions.

Non-IBM tools, such as JBuilder or Visual Cafe	Use the documentation for those products for installation and migration instructions.

Table 17. References for installation or migration information for application development software (continued)

Software:	Source of installation or migration information:
DB2 Universal Database Version 7.1	For more information about setting up DB2 and the implications for application programs, start with the *DB2 Universal Database... Quick Beginnings* book for your workstation platform.

3. **Recommendation:** If you are using VisualAge for Java, back up the workspace after you finish installing or upgrading VisualAge for Java. To do so, back up the `ide.icx` and `ide.ini` files.

4. Install or upgrade the appropriate assembly and deployment software on your workstation. For installation or migration instructions, see the following references:

Table 18. References for installation or migration information for assembly and deployment software

Software:	Source of installation or migration information:
WebSphere for z/OS Application Assembly tool	For workstation requirements and installation instructions, see “Steps for installing the Application Assembly tool” on page 136..
WebSphere for z/OS Administration application	For workstation requirements and installation instructions, see the topic on installing the Administration and Operations applications in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834.

5. Make sure your workstation’s environment variables are set correctly. The variables to check include CLASSPATH, JAVA_HOME, LIBPATH, and PATH. Depending on the products you installed, refer to the installation documentation for each product to determine whether these variables might be set automatically, or how to set them correctly.

6. If you are using the IBM WebSphere Test Environment feature of VisualAge for Java to run and test beans that require a persistent data store, make sure you specify the correct DB2 JDBC driver. JDBC V2.1 might not be the default for the DB2 product installation. Refer to the DB2 installation documentation for further details.

7. If you are using WebSphere Studio, make sure you complete the following steps after installing this tool:
   a. Start WebSphere Studio and complete the following steps:
      1) Select `File` → `New Project`, and name your project.
      2) Select `Project` → `VisualAge for Java` → `Install Studio Tools in VisualAge`
      3) Select `Project` → `Customize Publishing Stages`.
         In the dialog box that appears, enter `VAJ` for Stage Name, and click `Add` and then `OK`.
      4) Right-click on the VAJ stage icon in the right-hand pane, then select `Insert` and insert a server with the name `localhost:8080`

Result: A server icon appears for the default server. All the publishable resources in the project are automatically added to this server.

5) In the Publishing view, right-click on the `http://localhost:8080` server, select `Properties`, and click on `Define Publishing Targets`. Then set the paths to the WebSphere Test Environment document and servlets directories as follows:

```
html=D:\Program Files\IBM\VisualAge for Java\ide\
   project_resources\IBM WebSphere Test Environment\hosts\
   default_host\default_app\web

servlet=D:\Program Files\IBM\VisualAge for Java\ide\
   project_resources\IBM WebSphere Test Environment\hosts\
   default_host\default_app\servlets
```

b. Start VisualAge for Java and complete the following steps:
   - Select `Window → Options→ Remote Access To Tool API`.
   - Select the `Remote Access To Tool API` checkbox to activate remote access on VisualAge start-up.
   - If remote access is not currently running, click on `Start Remote Access To Tool API` to start it.
   - Select `Window → Options→ Resources`. Then add the following to the Workspace Classpath:

```
D:\Program Files\IBM\VisualAge for Java\ide\
    project_resources\IBM WebSphere Test Environment\hosts\
    default_host\default_app\servlets
```

c. Search for the `SERunner.properties` file. Open this properties file and make sure the `docRoot` and `serverRoot` paths are correct:

```
docRoot=D:\\Program Files\\IBM\\VisualAge for Java\\ide\\
    project_resources\\IBM WebSphere Test Environment\\hosts\\
    default_host\\default_app\\web

serverRoot=D:\\Program Files\\IBM\\VisualAge for Java\\ide\\
    project_resources\\IBM WebSphere Test Environment
```

8. If you are developing beans that require a persistent data store, create the DB2 tables that they will need. For more information about setting up DB2 tables, start with the *DB2 Universal Database... Quick Beginnings* book for your workstation platform.

9. If you plan to use the IBM Distributed Debugger and Object Level Trace to debug and trace your applications, see"Debugging and tracing distributed applications" on page 281 for more information.

Steps for setting up z/OS or OS/390

Almost all application development, assembly, and installation tasks take place on the workstation, but some tasks require a connection to the z/OS or OS/390 system on which you plan to deploy your J2EE application. When system programmers at your site install WebSphere for z/OS, they have the option of setting up the UNIX application development environment on z/OS or OS/390. The instructions here list minimum requirements, so you may verify the correct environment yourself. If necessary, see additional references for further details:

- See *z/OS UNIX System Services Planning*, GA22-7800 for information about setting up the UNIX environment.

- See *z/OS Communications Server: IP Configuration Guide*, SC31-8775 for information about setting up an FTP server. Use the same user ID and password that you will later use for the WebSphere for z/OS Administration application. (If necessary, see *WebSphere Application Server V4.0.1 for z/OS and OS/390: System Management User Interface*, SA22-7838.)

Perform the following steps to set up z/OS or OS/390:

1. Make sure you have TCP/IP connectivity between your workstation and the z/OS or OS/390 system on which WebSphere for z/OS resides. One way to check is to open a command prompt window and enter the ping command, specifying the TCP/IP host name.

2. On z/OS or OS/390 UNIX, check each application developer's region size (MAXASSIZE in BPXPRMxx or ASSIZEMAX on the RACF ADDUSER or ALTUSER commands). The rule of thumb is to run with the largest region size possible, but start with a minimum size of 256 MB. The size can be limited by the IEFUSI exit, JES2 EXIT06, JES3 IATUX03, or TSO segment defaults. If the compiler runs out of memory, you may need to increase the application developer's region size.

3. On z/OS or OS/390, set up an FTP server that has write access to the HFS.

Chapter 6. Creating new application components to be installed in a J2EE server

Once you have installed or upgraded the appropriate workstation software, as noted in Chapter 5, "Setting up the application development environment," on page 109, you are ready to create J2EE application components. Because documentation for developing J2EE application components is available through other WebSphere, Sun Microsystems, and third-party documents, the following topics list only the rules or guidelines to keep in mind while you are coding application components for installation in an EJB container or Web container in a WebSphere for z/OS J2EE server.

When you develop an Enterprise bean or Web component, you may concentrate solely on the business and application logic for each component. The J2EE server's Web and EJB containers are designed to handle transactions, security, and scalability related to access to any Enterprise Information Systems (for example, DB2 databases; Enterprise Resource Planning systems; mainframe systems such as CICS and IMS; RDBMs; and legacy applications).

This topic addresses the guidelines for developing and testing only the types of components that may be installed in the J2EE server:

- Enterprise beans written to the Sun Microsystems Enterprise JavaBeans (EJB) 1.1 and 1.0 specifications, and
- Web applications written to the Sun Microsystems Java Servlet Specification, V2.2.

For information about developing J2EE application clients, see Chapter 9, "Creating and running WebSphere for z/OS client applications," on page 217.

Creating Enterprise beans

The WebSphere for z/OS J2EE server supports all types of Enterprise JavaBeans:

- Stateless session
- Stateful session
- Container-managed (CMP) entity
- Bean-managed (BMP) entity

If you are not familiar with these terms, or with the concepts and requirements for coding Enterprise beans to the Sun Microsystems EJB specifications, you may use either IBM or non-IBM resources for information about the bean programming model, application programming interfaces, and services that you may use. Make sure that you also note the temporary limitations that are specific to z/OS and OS/390, which are covered in "Developing Enterprise beans" on page 117.

Perhaps the easiest approach to developing beans is to use the IBM WebSphere Studio Application Developer product, which is an easy-to-use, integrated development environment for building, testing, and deploying J2EE™ applications with HTML pages, servlets, JavaServer™ Page (JSP), and Enterprise JavaBean™ (EJB™) components. This product enables developers to fully test entity and session beans, including JNDI lookups, remote method calls, and method calls on the home interface.

© Copyright IBM Corp. 2000, 2003

Recommendation: Although you can test Enterprise beans on the workstation, you also should test them on the z/OS or OS/390 platform as well, because there may be subtle differences in the run-time environment. For example, if your beans use DB2, differences between DB2 support on the workstation and on z/OS or OS/390 might become evident. Use the guidelines in this chapter to identify potential differences between the workstation and z/OS or OS/390 run-time environments.

Checklist for developing Enterprise beans

Use the following checklist to make sure you have completed all of the necessary tasks related to creating an Enterprise bean.

Before you begin:

- For further details about this development process, use the following resources:
  - The Preventive Service Planning (PSP) bucket for WebSphere for z/OS.
  - The Sun Microsystems EJB 1.1 or 1.0 specification document, or an equivalent information source.
  - Instructions for using the application development tools you have selected. These instructions appear in product documentation, which is available through the IBM home page (`www.ibm.com`) by clicking on the links for software products. For example, to find the documentation for using the IBM WebSphere Studio Application Developer product, follow these steps after the IBM home page loads into your browser:
    1. Click on the following links: `Products → Software → WebSphere → Products → Application development`.
    2. Using the Select a Product drop-down, select WebSphere Studio Application Developer.
    3. Click on library for links to all of the product documentation.

Table 19. Checklist for developing Enterprise beans

Check when completed:	**Task:**
☐	Add a new project and package for the beans you are creating **Note:** A project is a concept used in WebSphere Studio Application Developer..
☐	Add an Enterprise bean, specifying a name and type
☐	Define the bean class attributes and interfaces
☐	Add import statements as necessary (for example, associated bean classes and Java classes for specific functions)
☐	Code business logic methods for bean classes
☐	**Tips** for creating CMP beans: • Allow the tool to generate finder helper interface to support finder methods • Define which bean attributes require container-managed persistence • Import classes required for data access (for example, db2long JDBC classes) • Import a database schema, specifying the connection type and data source • Generate a map for the appropriate table from the database schema • Map bean attributes to the associated column name in table maps
☐	**Tips** for creating BMP beans: • Code the methods that manage the bean's lifecycle • If you are using dynamic SQL, make sure that #sql statements include the high-level qualifier that matches the one used when creating db2long tables
☐	Add properties, including: • The JNDI name for the BeanHome • Transaction attribute

Table 19. Checklist for developing Enterprise beans (continued)

Check when completed:	Task:
☐	Generate deployed code for the bean **Note:** For CMP beans, you need to generate stubs, ties, and persisters.
☐	Test application components
☐	Package the beans in JAR files **Tip:** See "Preparing applications for assembly and installation" on page 120

Developing Enterprise beans

As you develop Enterprise beans to install in a WebSphere for z/OS J2EE server, keep the following points in mind:

Rules:

- If your bean manages transactions (in other words, has a "bean-managed transactions" attribute), you need to be aware these rules and ways in which WebSphere for z/OS resolves uncommitted transactions:
  - Your bean cannot start a global transaction until it resolves any uncommitted local transactions or resources that it may have.
  - In a global transaction, if the bean finishes its processing without first resolving its transaction, WebSphere for z/OS will roll back the bean's transaction.
  - If the bean finishes its processing without first resolving any resource-manager local transactions, WebSphere for z/OS will roll back those transactions.
- WebSphere for z/OS supports role-based security for Enterprise beans. You may use the `isCallerInRole` and `getCallerPrincipal` methods if you want to code beans to perform security checks.

  A role name cannot contain blanks, and cannot exceed 246 characters. Role names, however, may be in mixed case.

Limitation:

WebSphere for z/OS rejects inbound or outbound requests (GIOP messages) that exceed 10 megabytes in size. If you design an application component to send any type of GIOP message, you need to do the following:

- Calculate the size of the entire message, including headers and other parameters that you intend to pass. The entire size of a transmitted message cannot exceed 10MB.
- Decide how to handle the `CORBA::NO_MEMORY` exception that WebSphere for z/OS issues when it detects messages that exceed the 10MB size limitation. One possibility is logging the error.

Note: This 10MB limit is the result of WebSphere for z/OS implementation; it is not an architected or programming model restriction.

Guidelines:

- You may code your beans to use the JDBC application programming interface (API) to access db2long data. For additional information, see:
  - *DB2 Application Programming and SQL Guide*, SC26-9933 for instructions on using the db2long for z/OS or OS/390 JDBC driver.

  - `http://java.sun.com/products.jdbc` for detailed information about the JDBC API.
- If the beans you code require db2long to store persistent data, you need to use the type 2 JDBC driver that ships with DB2 Universal Database for z/OS and OS/390 Version 7.1. Given the use of this driver, you may design your bean to:
  - Participate in either global or local transactions.
  - Have multiple connections with db2long. (Note, however, that performance is better with only one connection.)
  - Assign a different outcome for each db2long connection, when the bean does not run under a global transaction.
  - If the beans you code are Stateful Session EJBs, and, based on WebSphere for z/OS extended deployment descriptors, require the session data to be stored in memory, you must disable the Server Region Recycle Function by setting the Server Region Recycle Limit to 0 (zero). Further, use of the z/OS WLM operator command **V WLM,APPLENV=**<generic server name>,REFRESH is discouraged because existing server regions may never terminate.
- If you supply a custom finder for a bean that uses container-managed persistence (CMP bean), check the SQL `SELECT` statement in the custom finder. If the `SELECT` statement contains the `FOR UPDATE` clause, or the keyword `ORDER BY` or `DISTINCT`, you must use one of the following settings to avoid encountering an SQL error (`SQLCODE` -126) when the J2EE server attempts to run the CMP bean:
  - Specify the `ReadOnly` setting on the method level, through the appropriate application development or assembly tool;
  - Specify env-entry `com.ibm.websphere.persistence.bean.managed.custom.finder.access.intent` as true in the bean's standard deployment descriptor (*ejb*`-jar.xml`); or
  - Specify the property `com.ibm.websphere.persistence.bean.managed.custom.finder.access.intent=1` in the JVM properties file for the J2EE server. Using this property in the J2EE server's JVM property file sets the same access intent for all CMP beans that run in the server.

  For additional information, see
  - "Isolating transactions that access persistent data" on page 56 and "Controlling concurrent access to persistent data" on page 57 for IBM extensions related to access intent for CMP beans.
  - "JVM properties and properties files" on page 339 for the `com.ibm.websphere.persistence.bean.managed.custom.finder.access.intent` property.

Creating Web applications

A Web application can include one or more of any of the following Web components:
- Servlets
- JavaServer Pages (JSPs)
- Utility classes
- Static documents

The roles each Web component should play in a Web application are defined in the Java™ Servlet Specification, V2.2, which is available at:

```
http://www.javasoft.com
```

Developing Web components

An instance of a Web application exists within a single WebSphere for z/OS J2EE server. It uses servlet context to obtain references to other local objects and to share data with other applications. A servlet can be replicated if it is marked distributable. In this case, however, you must ensure that one of the following conditions is true:

- Either the servlet is not written to leave objects around for later processing, or
- The servlet is installed in a J2EE server that cannot be replicated within a sysplex.

Before a Web application can be installed on a J2EE server:

1. All of the components of the Web application must be packaged into a Web application archive (WAR) file. This file has a file extension of .war, and must include a web.xml file. (The web.xml file indicates how the application will be served when requested by a client.) If you use WebSphere Studio Application Developer to develop your Web application, you can also use it to package all of the components into a JAR file.
2. This WAR file must then be packaged as part of an Enterprise archive (EAR) file. (See "Steps for assembling a new J2EE application" on page 137 for information on how to perform this packaging.) An EAR file is an archive file similar to a JAR file, with a specific directory structure and format and has an extension of .ear. It includes an application.xml file which contains the descriptive meta information which ties together all of the WAR or EJB JAR files packaged in the EAR file.

It is the responsibility of the Application Component Provider to write the business and application logic for his application. An Application Component Provider can rely on the Web and EJB containers to handle transactions, security, and scalability related to Enterprise Information Systems (EIS) access. (EISs include DB2 databases, Enterprise Resource Planning systems, mainframe systems such as CICS and IMS, RDBMS, and legacy applications.)

Limitation: If your Web application uses RMI/IIOP for communication with other components in a J2EE server, you need to be aware that WebSphere for z/OS rejects inbound or outbound requests (GIOP messages) that exceed 10 megabytes in size. If you design an application component to send any type of GIOP message, you need to do the following:

- Calculate the size of the entire message, including headers and other parameters that you intend to pass. The entire size of a transmitted message cannot exceed 10MB.
- Decide how to handle the `CORBA::NO_MEMORY` exception that WebSphere for z/OS issues when it detects messages that exceed the 10MB size limitation. One possibility is logging the error.

Note: This 10MB limit is the result of WebSphere for z/OS implementation; it is not an architected or programming model restriction.

Guideline: WebSphere for z/OS supports role-based security for servlets. You may use the `isCallerInRole` and `getCallerPrincipal` methods if you want to code servlets to perform security checks. **Rule:** A role name cannot contain blanks, and cannot exceed 246 characters. Role names, however, may be in mixed case.

Preparing applications for assembly and installation

Before you can install a J2EE application in a WebSphere for z/OS J2EE server, you need to package the JAR or WAR files for individual components together into an Enterprise archive (EAR) file. An EAR file is an archive file similar to a JAR file, with a specific directory structure and format, and has an extension of `.ear`. To create an EAR file for your application, you may use one of the IBM WebSphere Studio Application Developer editions or the WebSphere for z/OS Application Assembly tool.

During this assembly process, you not only define the individual components of a single application, but also deploy those components for the WebSphere for z/OS environment. WebSphere Studio Application Developer and the Application Assembly tool both generate an application.xml file (part of the EAR file contents), which contains the descriptive meta-information that ties together all of the JAR or WAR files that you might package in a single application. This metadata enables the J2EE server to understand the content and individual component requirements of an installed J2EE application.

Also as part of the development and assembly processes, you can define classpaths for application components. For example, you can use the Application Assembly tool to set or modify the Manifest classpath for both Enterprise beans and Web components. When your application is installed in a J2EE server, WebSphere for z/OS class loaders use these classpaths to find and load application classes for execution. WebSphere for z/OS uses several different class loaders to install different application components in a J2EE server. The interaction of these class loaders can affect the packaging scheme you use, as well as affect the successful execution of your applications in the WebSphere for z/OS run-time environment.

You can successfully use WebSphere for z/OS classloader defaults for most applications. If you encounter classloader errors, however, you might need to alter classloader operation or repackage application components. To do so, you need to understand how the class loaders interact, and how their operation can be altered.

The following table shows the associated information for preparing application components for assembly and installation:

If you want to...	Associated information (See . . .)
Learn about WebSphere for z/OS classloader operation and how to change it	• "Overview of WebSphere for z/OS classloader operation" • "Guidelines for setting classloader mode and application packaging" on page 131 • "Steps for setting trace options for classloader operation" on page 133
To assemble and deploy applications	Topics in Chapter 7, "Assembling a J2EE application," on page 135

Overview of WebSphere for z/OS classloader operation

When WebSphere for z/OS receives requests for an Enterprise bean or servlet installed in a J2EE server, WebSphere for z/OS calls on its class loaders to find and load the appropriate application classes for execution. The types of class loaders

WebSphere for z/OS uses depend on the classloader mode in effect for the J2EE server. WebSphere for z/OS supports the following classloader modes (which are also known as “visibility modes”):

- Application mode (which is the default classloader mode)
- Compatibility mode
- Server mode
- Module mode
- J2EE Application mode

You can successfully use WebSphere for z/OS classloader defaults for most applications. You might, however, encounter classloader errors in the run-time environment because the interaction of these class loaders can affect the successful execution of your applications. If you do encounter errors, you might need to alter classloader operation or repackage application components.

Recommendations:

- IBM recommends using the default classloader mode (application mode) for most applications.
- If you plan to change the classloader mode, consider testing your applications in a test server with the new classloader setting. To more easily diagnose classloader operation, enable tracing for the J2EE server as instructed in “Steps for setting trace options for classloader operation” on page 133.
- If you encounter classloader errors that you cannot resolve with the suggested diagnostic procedures, consider reviewing Table 20 in “Guidelines for setting classloader mode and application packaging” on page 131 to determine whether a different classloader mode is appropriate for your application.
- Consider testing your applications in a J2EE server that has its classloader mode set to `J2EE Application mode`. This mode complies with the Sun Microsystems J2EE 1.3 specification for classloader operation.

A class loader is a class that performs the function of loading a named class or interface. When an application client requests an Enterprise bean, for example, the WebSphere for z/OS run-time creates a class loader to find and load the classes from the appropriate JAR module. Each of the WebSphere for z/OS class loaders share the following common features, but each has different values that define the class loader and its behavior:

Context classpath
Each class loader has an associated classpath that is defined for the specific type of class loader, and for the module (JAR or WAR) associated with that class loader. The classpath defines the part of the HFS file system that the class loader searches to locate a requested class.

Delegation mode
Each class loader has an associated parent class loader and a delegation mode. The delegation mode determines when the class loader will delegate a load request to its parent; the class loader may delegate to its parent either before or after searching its own classpath.

To understand these concepts more fully, look at Figure 20 on page 122, which illustrates the default classloader mode for WebSphere for z/OS. In the illustration, the lightly outlined boxes (at the top) indicate the class loaders that are always part of the classloader hierarchy, or family tree, regardless of the classloader mode in effect. It is important to note that parent class loaders cannot “see” the classes in the classpaths of child class loaders.

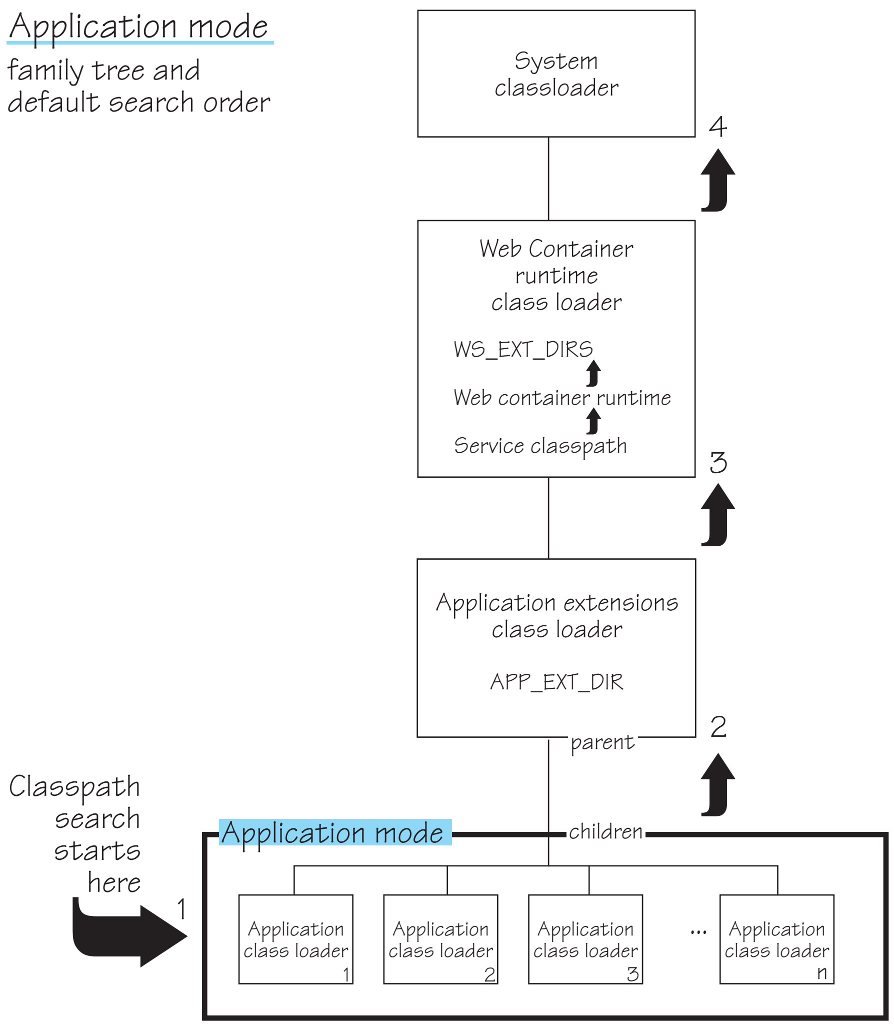

Application mode
family tree and default search order
System classloader
4
Web Container runtime class loader
WS_EXT_DIRS
Web container runtime
Service classpath
3
Application extensions class loader
APP_EXT_DIR
parent
2
Classpath search starts here
1
Application mode
children
Application class loader 1
Application class loader 2
Application class loader 3
...
Application class loader n

Figure 20. WebSphere for z/OS default classloader mode, types, and search order

As shown in Figure 20, application mode dictates the following conditions:

- WebSphere for z/OS creates one application class loader for each application installed in the J2EE server. These class loaders are shown in the heavily outlined box at the bottom of the illustration. The classpath for a given class loader includes the module paths for all WAR or JAR files within the application.
- With all defaults in effect, the search for a particular class begins with the application class loader associated with the application containing that class. If the class is not found in that application's modules, the application class loader passes the search request to its parent. The search request progresses up the family tree until the class is found. The search order is illustrated by arrows at the bottom left and on the right of the illustration.

  The delegation mode for these application class loaders may be altered, as shown in Figure 23 in "Changing the default search order for classes" on page 126..
- The Application extensions class loader is the parent of all application class loaders. The classpath for the Application extensions class loader consists of the path specified on the `APP_EXT_DIR` environment variable for the J2EE server. This variable is intended for classes that can be shared by all applications installed in the J2EE server. The default value for this variable is:

`*CBCONFIG*/apps/*SRVNAME*/app`
where

CBCONFIG
Is a read/write directory that you specify at installation time as the directory into which WebSphere for z/OS is to write configuration data and environment files. The default is `/WebSphere390/CB390`.

SRVNAME
Is the generic server name.

The delegation mode for the Application extensions class loader cannot be changed; this class loader always searches its own classpath before delegating to its parent.

- The Web Container run-time class loader is the parent of the Application extensions class loader. The classpath for the Web Container run-time class loader contains three parts:
  – The Service classpath, which is defined by the JVM property `com.ibm.ws390.wc.SERVICE.classpath` for the J2EE server. This property is reserved for service.
  – The Web Container run-time classpath, which is built by WebSphere for z/OS based on a list of JAR files, each of which is concatenated with the Web Container install root directory.
  – The WebSphere extensions classpath, which is defined by the `WS_EXT_DIRS` environment variable for the J2EE server. This variable is intended for classes that extend the function of the J2EE server.

  The delegation mode for the Web Container run-time class loader cannot be changed; this class loader always searches its own classpaths before delegating to its parent. The Web Container run-time class loader begins its search for a particular class with the Service classpath, then the Web Container run-time classpath, and finally the `WS_EXT_DIRS` classpath. This search sequence cannot be changed.
- The System class loader is the parent of the Web Container run-time class loader. After searching its own classpath, the System class loader may pass the search request to another class loader, depending on the delegation mode in effect. For an example of this delegation with application mode, see Figure 23 on page 127.

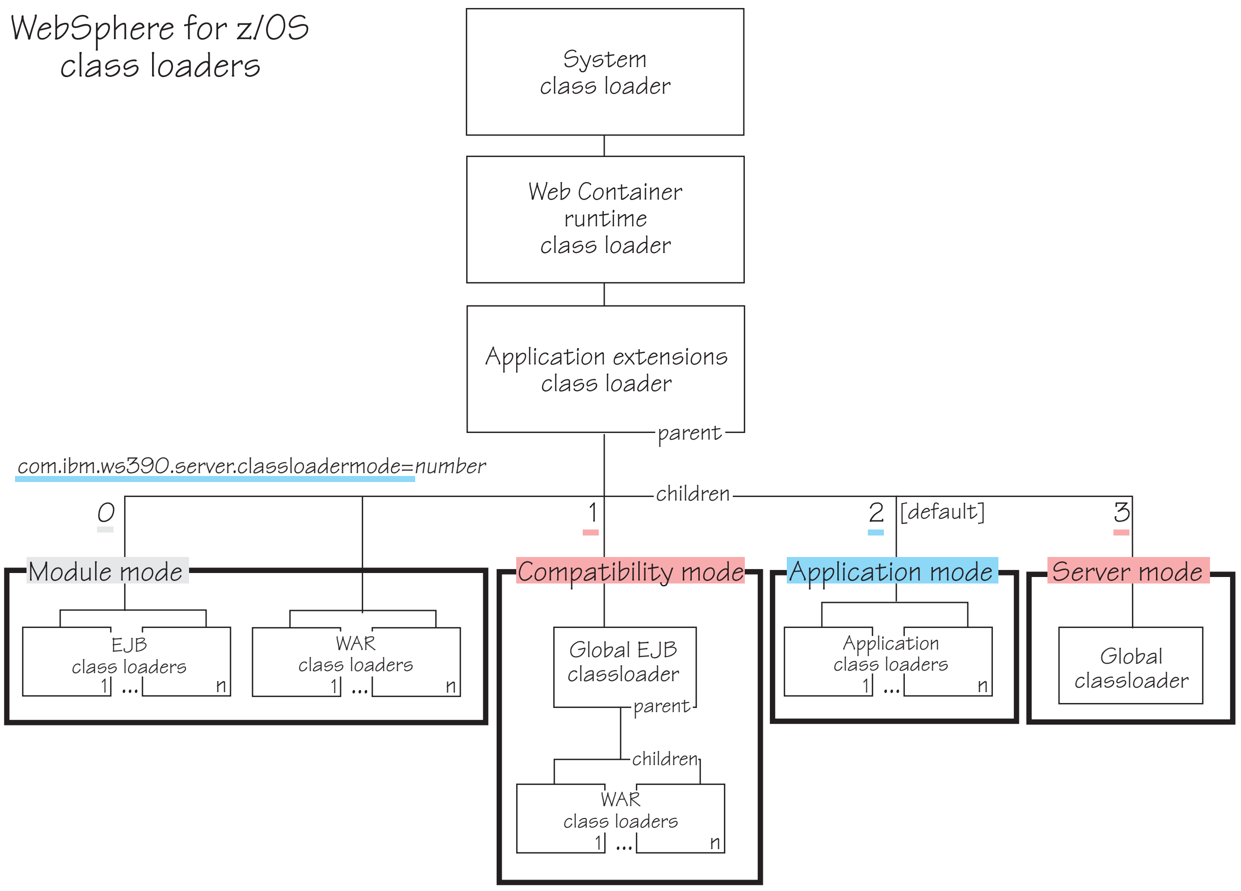
Setting alternative classloader modes

As noted in the description of application mode, the lightly outlined boxes at the top of Figure 20 on page 122 indicate the class loaders that are always part of the classloader family tree, regardless of the classloader mode in effect. When you change the classloader mode, however, you alter the classloaders in the heavily lined box. In other words, you replace the application class loaders with other types. WebSphere for z/OS supports five classloader modes:

- The default mode and three alternatives, which are set through environment variable `com.ibm.ws390.server.classloadermode`. These modes are shown in Figure 21 on page 124.
- A classloader mode designed for compliance with the J2EE 1.3 specification. This mode is set through the JVM property `com.ibm.ws.classloader.J2EEApplicationMode`, and illustrated in Figure 22 on page 125.

Recommendation: Use the WebSphere for z/OS Administration application to set `com.ibm.ws390.server.classloadermode` by specifying its value on the properties form for a J2EE server. This setting is stored in the `current.env` file for the J2EE server, which means the value applies for all instances of the J2EE server. You can override the value for only a given server instance by setting `com.ibm.ws390.server.classloadermode` in a JVM properties file.

Reviewing Table 20 in "Guidelines for setting classloader mode and application packaging" on page 131 can help you determine which classloader mode is appropriate for your application.



WebSphere for z/OS class loaders
System class loader
Web Container runtime class loader
Application extensions class loader
parent
com.ibm.ws390.server.classloadermode=number
children
0
1
2 [default]
3
Module mode
EJB class loaders 1 ... n
WAR class loaders 1 ... n
Compatibility mode
Global EJB classloader
parent
children
WAR class loaders 1 ... n
Application mode
Application class loaders 1 ... n
Server mode
Global classloader

Figure 21. WebSphere for z/OS classloader "family tree" with alternative modes

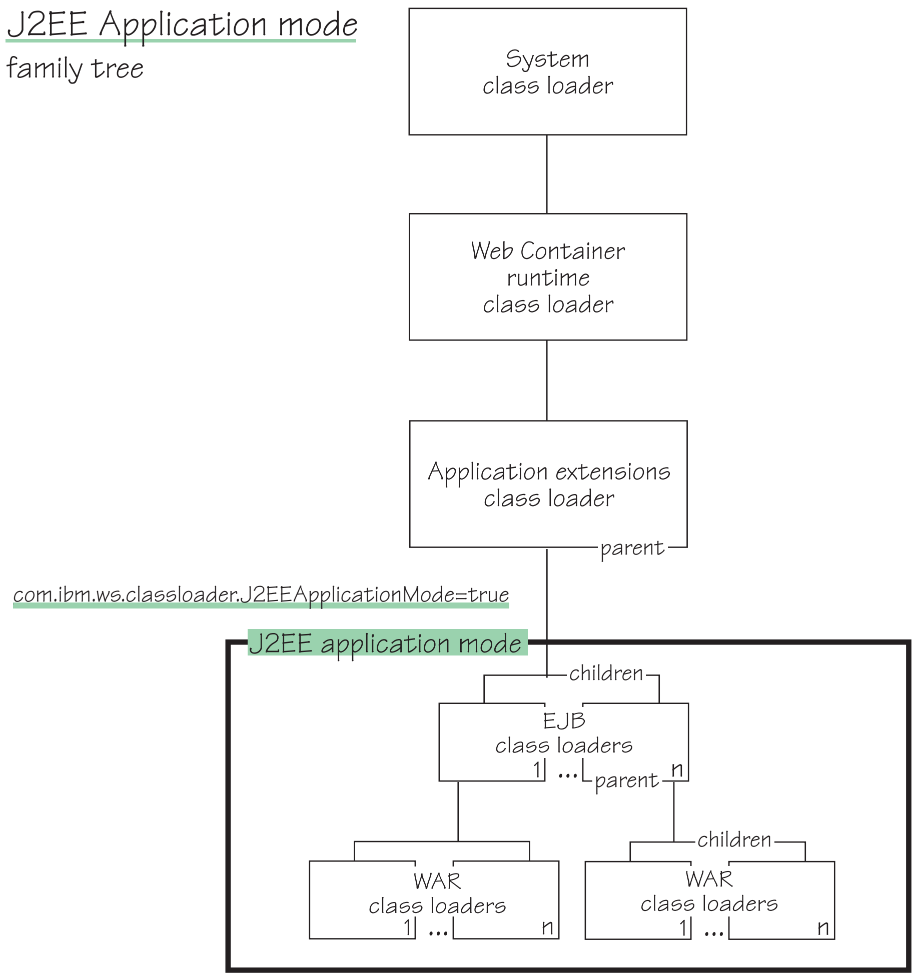


Figure 22. WebSphere for z/OS classloader mode for J2EE 1.3 compliance: "family tree"

The alternative classloader modes illustrated in Figure 21 on page 124 and Figure 22 introduce the following additional classloader types:

- EJB class loaders, one for each EJB module within a given application installed in the J2EE server. The classpath for an EJB class loader consists of two parts:
  - EJB JAR file (the class loader searches the JAR file that contains the EJB classes for the Enterprise bean module)
  - MANIFEST classpath, if the JAR file includes a classpath specification with references to JAR files or directories that contain bare class files or resource files, or both. All such references should be specified using paths that are relative to the directory of the given application.
- Global EJB class loader, which has the same classpath as an EJB class loader, but has a server-wide scope. In other words, only one Global EJB class loader handles requests for all of the Enterprise bean modules in all applications installed in the J2EE server.
- WAR class loaders, one for each WAR module within a given application installed in the J2EE server. The classpath for a WAR class loader consists of two parts:
  - `WEB-INF/classes`, the directory within the WAR module that contains bare classes.
  - `WEB-INF/lib`, the directory within the WAR module that contains JAR files that provide classes used by the servlets packaged within the WAR module.

  – MANIFEST classpath, if the JAR file includes a classpath specification with references to JAR files or directories that contain bare class files or resource files, or both. All such references should be specified using paths that are relative to the directory of the given application.
- Global class loader, which handles all types of application modules, for all applications installed in the J2EE server.

For the alternative classloader modes, one or more JVM property values determine the default delegation for particular class loaders. The default delegation for each classloader mode may be altered, as described in "Changing the default search order for classes."

Changing the default search order for classes

The following series of figures illustrates each classloader mode hierarchy, along with the JVM properties you may use to alter the default search path.

Figure 23 on page 127 illustrates the classloader hierarchy with application mode. With application mode, the value of JVM property `com.ibm.ws.classloader.warDelegationMode` controls the delegation behavior for the application class loaders. When it receives a search request for a specific class, WebSphere for z/OS passes the request to the appropriate application class loader. Depending on the delegation mode, the application class loader behaves in one of two ways:

- With `com.ibm.ws.classloader.warDelegationMode` set to `false` (the default), the application class loader searches its own classpath for the class. If it does not find the class, the class loader passes the search request to its parent.
- With `com.ibm.ws.classloader.warDelegationMode` set to `true`, the application class loader immediately delegates the search request to its parent class loader. So the parent class loader's classpath is the first one searched for the requested class. The search request progresses up the family tree until the class is found. If the requested class is not found by any class loader in the hierarchy, the System class loader passes the request to the application class loader; only at this point is the application class loader's classpath searched for the requested class.

The rest of the information for this figure, such as the class loaders in the family tree, is the same as discussed for Figure 20 on page 122.

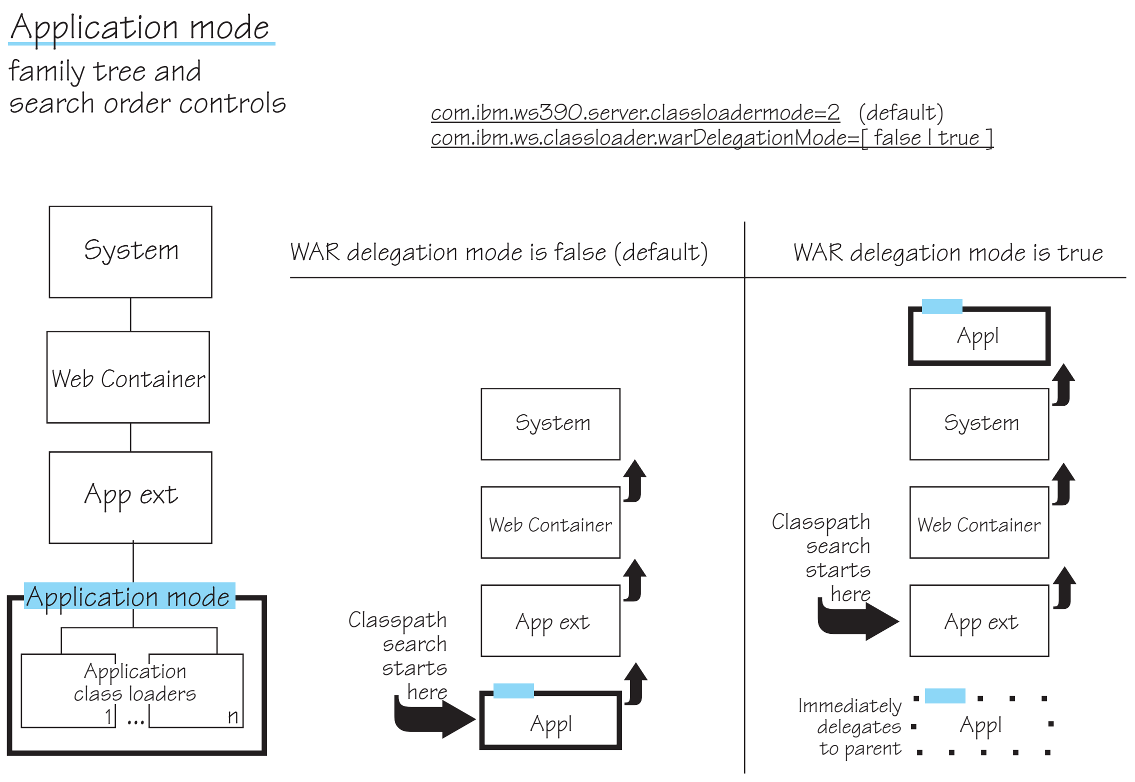

Application mode
family tree and
search order controls
com.ibm.ws390.server.classloadermode=2 (default)
com.ibm.ws.classloader.warDelegationMode=[false | true]
System
Web Container
App ext
Application mode
Application class loaders
1 ... n
WAR delegation mode is false (default)
System
Web Container
App ext
Appl
Classpath search starts here
WAR delegation mode is true
Appl
System
Web Container
App ext
Classpath search starts here
Immediately delegates to parent
Appl

Figure 23. Settings for changing the search order (delegation) for application mode

Figure 24 on page 128 illustrates the classloader hierarchy with compatibility mode. With compatibility mode:

- There are zero or more WAR class loaders, depending on the number of WAR modules in the applications installed in the J2EE server. Each class loader handles a single WAR module. With compatibility mode, the value of JVM property `com.ibm.ws.classloader.warDelegationMode` controls the delegation behavior for the WAR class loaders; the default delegation is false (search own classpath before delegating to parent).
- The Global EJB class loader is the parent of all of the WAR class loaders. With compatibility mode, the value of JVM property `com.ibm.ws.classloader.ejbDelegationMode` controls the delegation behavior for the Global EJB class loader; the default delegation is true (immediately delegate to parent).
- The Application extensions class loader (abbreviated as “App ext” in the figure) is the parent of the Global EJB class loader.
- The remainder of the class loader hierarchy is the same as described for Figure 20 on page 122.

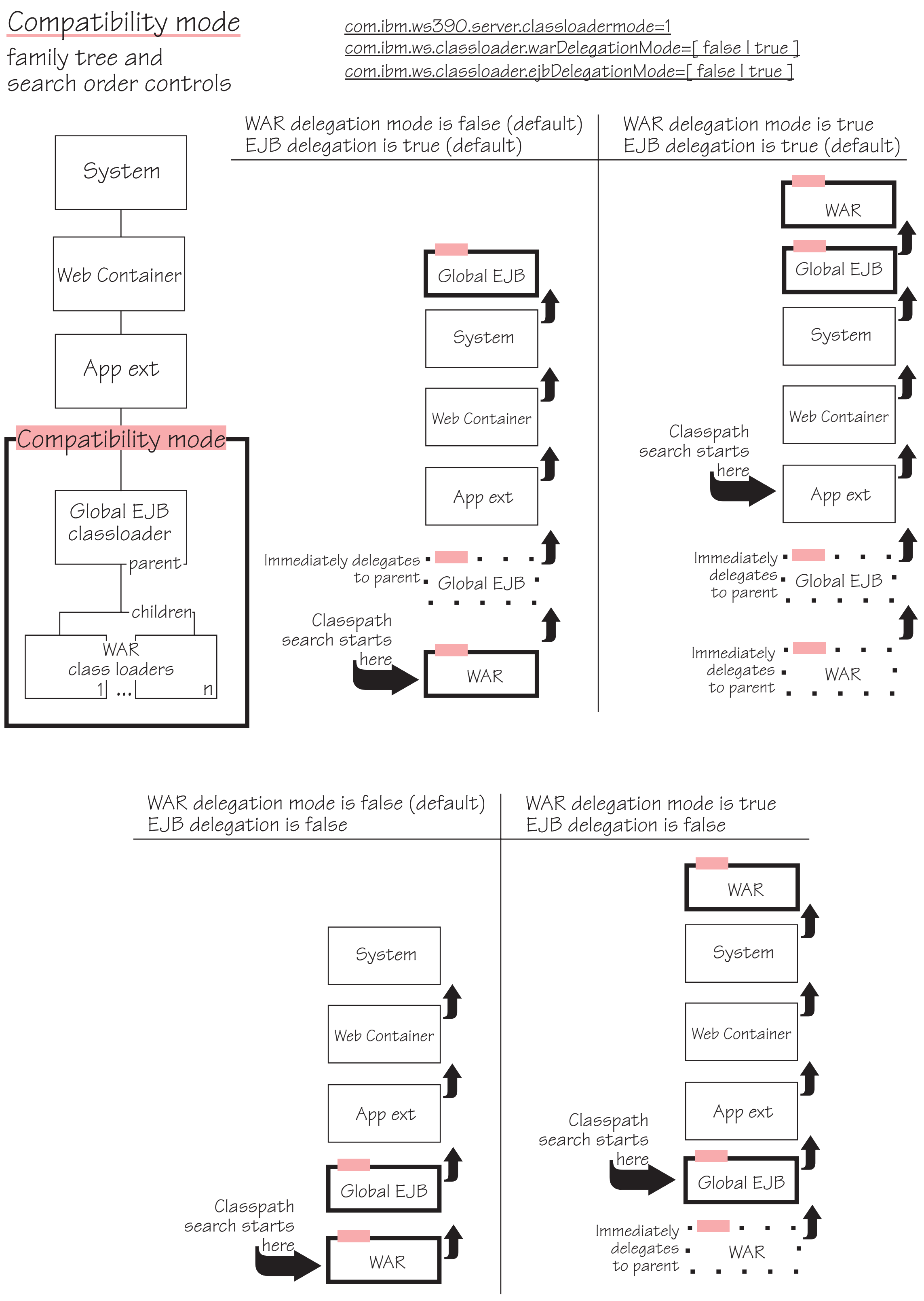
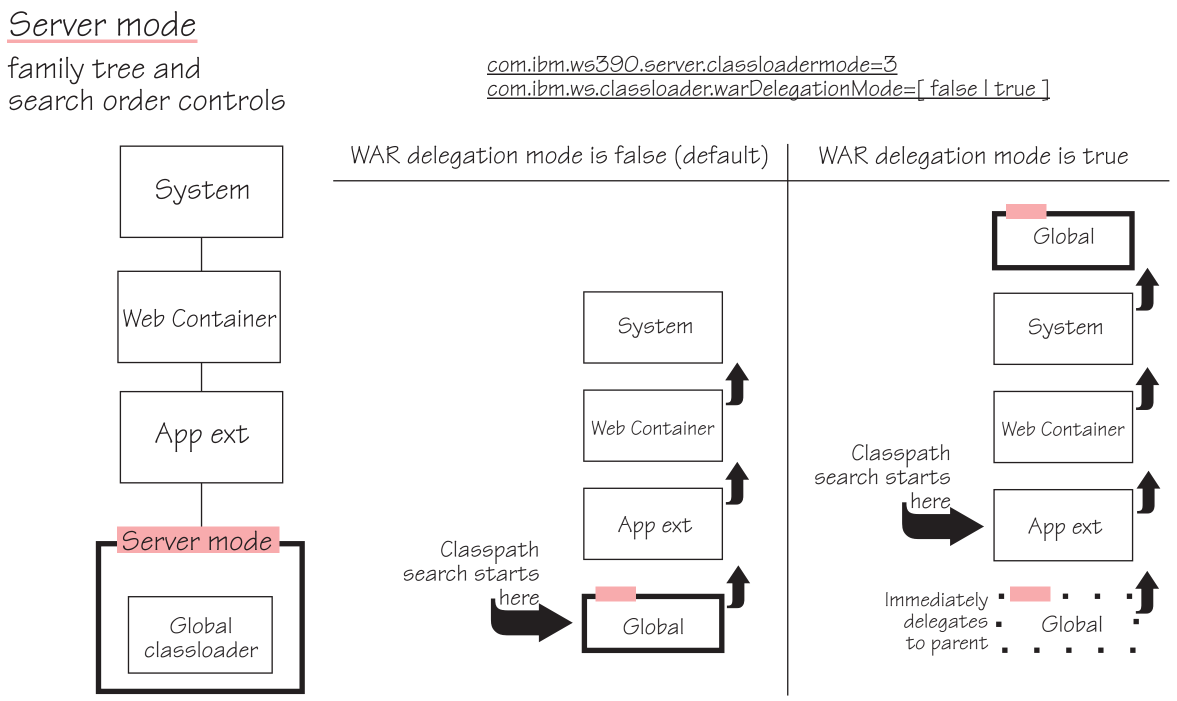

Compatibility mode
family tree and
search order controls
com.ibm.ws390.server.classloadermode=1
com.ibm.ws.classloader.warDelegationMode=[false | true]
com.ibm.ws.classloader.ejbDelegationMode=[false | true]
System
Web Container
App ext
Compatibility mode
Global EJB classloader
parent
children
WAR class loaders
1 ... n
WAR delegation mode is false (default)
EJB delegation is true (default)
Global EJB
System
Web Container
App ext
Immediately delegates to parent
Global EJB
Classpath search starts here
WAR
WAR delegation mode is true
EJB delegation is true (default)
WAR
Global EJB
System
Web Container
Classpath search starts here
App ext
Immediately delegates to parent
Global EJB
Immediately delegates to parent
WAR
WAR delegation mode is false (default)
EJB delegation is false
System
Web Container
App ext
Global EJB
Classpath search starts here
WAR
WAR delegation mode is true
EJB delegation is false
WAR
System
Web Container
App ext
Classpath search starts here
Global EJB
Immediately delegates to parent
WAR

Figure 24. Settings for changing the search order (delegation) for compatibility mode

Figure 25 on page 129 illustrates the classloader hierarchy with server mode. With server mode:

- The Global class loader handles both WAR and EJB modules. The one class loader handles all modules for all applications installed in the J2EE server. With server mode, the value of JVM property `com.ibm.ws.classloader.warDelegationMode` controls the delegation behavior for the Global class loader; the default delegation is false (search own classpath before delegating to parent).
- The Application extensions class loader (abbreviated as "App ext" in the figure) is the parent of the Global class loader.
- The remainder of the class loader hierarchy is the same as described for Figure 20 on page 122.



Server mode
family tree and search order controls
com.ibm.ws390.server.classloadermode=3
com.ibm.ws.classloader.warDelegationMode=[false | true]
WAR delegation mode is false (default)
WAR delegation mode is true
System
Web Container
App ext
Server mode
Global classloader
Classpath search starts here
Global
Immediately delegates to parent

Figure 25. Settings for changing the search order (delegation) for server mode

Figure 26 on page 130 illustrates the classloader hierarchy with module mode. With module mode:

- There are zero or more WAR class loaders, depending on the number of WAR modules in the applications installed in the J2EE server. Each class loader handles a single WAR module. With module mode, the value of JVM property `com.ibm.ws.classloader.warDelegationMode` controls the delegation behavior for the WAR class loaders; the default delegation is false (search own classpath before delegating to parent).
- There are zero or more EJB class loaders, depending on the number of EJB modules in the applications installed in the J2EE server. Each class loader handles a single EJB module. With module mode, the value of JVM property `com.ibm.ws.classloader.ejbDelegationMode` controls the delegation behavior for the EJB class loaders; the default delegation is true (immediately delegate to parent).
- The Application extensions class loader (abbreviated as "App ext" in the figure) is the parent of all of the WAR and EJB class loaders.
- The remainder of the class loader hierarchy is the same as described for Figure 20 on page 122.

Recommendation: If you are currently using module mode, which is a deprecated value for `com.ibm.ws390.server.classloadermode`, IBM recommends changing this setting to 2 (application mode), which is the default mode.

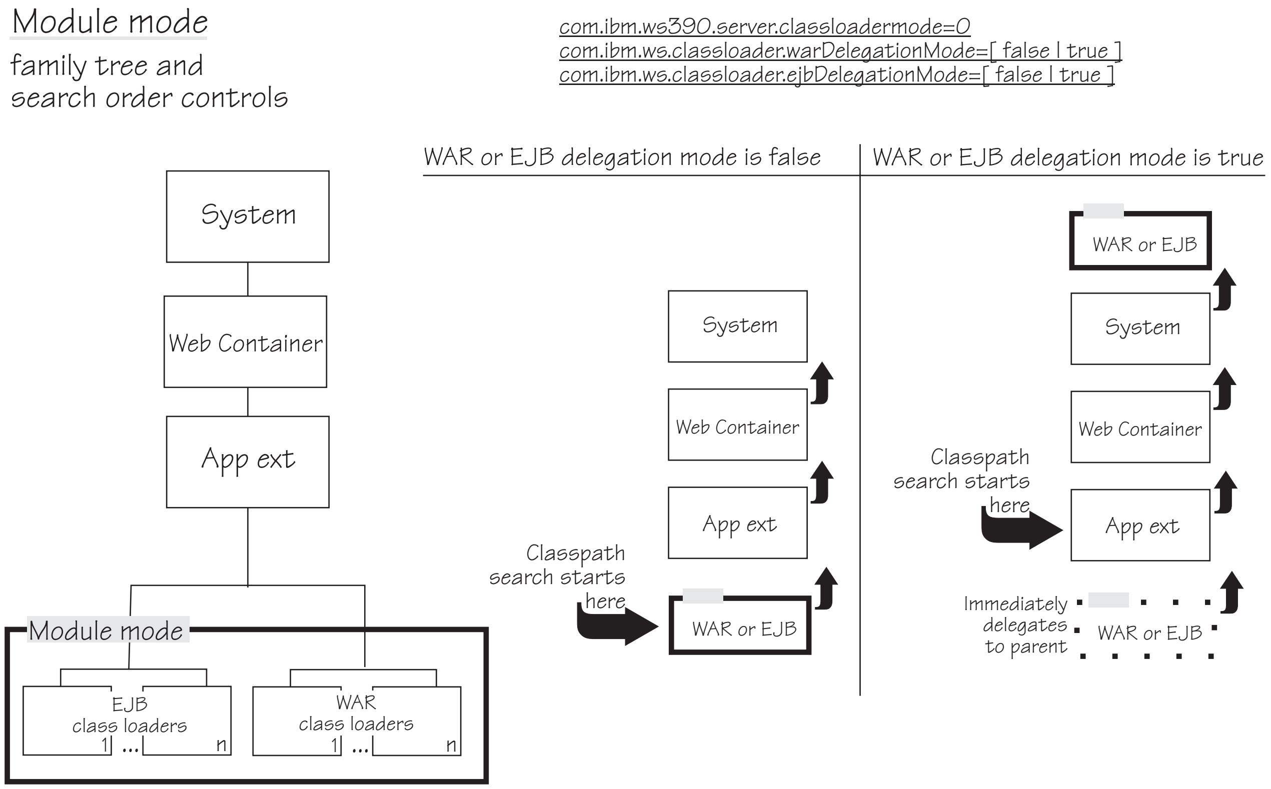

Module mode
family tree and
search order controls
com.ibm.ws390.server.classloadermode=0
com.ibm.ws.classloader.warDelegationMode=[false | true]
com.ibm.ws.classloader.ejbDelegationMode=[false | true]
WAR or EJB delegation mode is false
WAR or EJB delegation mode is true
System
Web Container
App ext
Module mode
EJB
class loaders
1 ... n
WAR
class loaders
1 ... n
Classpath
search starts
here
WAR or EJB
Immediately
delegates
to parent

Figure 26. Settings for changing the search order (delegation) for module mode

Figure 27 on page 131 illustrates the classloader hierarchy with J2EE Application mode. With J2EE Application mode:

- There are zero or more WAR class loaders, depending on the number of WAR modules in the applications installed in the J2EE server. Each class loader handles a single WAR module. With J2EE Application mode, the value of JVM property `com.ibm.ws.classloader.warDelegationMode` controls the delegation behavior for the WAR class loaders; the default delegation is true (immediately delegate to parent).
- There is one EJB class loader for each application installed in the J2EE server. Each EJB class loader is the parent of the WAR class loaders associated with the WAR modules within a single application. With J2EE Application mode, the value of JVM property `com.ibm.ws.classloader.warDelegationMode` controls the delegation behavior for the EJB class loaders; the default delegation is true (immediately delegate to parent).
- The Application extensions class loader (abbreviated as “App ext” in the figure) is the parent of all of the application class loaders.
- The remainder of the class loader hierarchy is the same as described for Figure 20 on page 122.

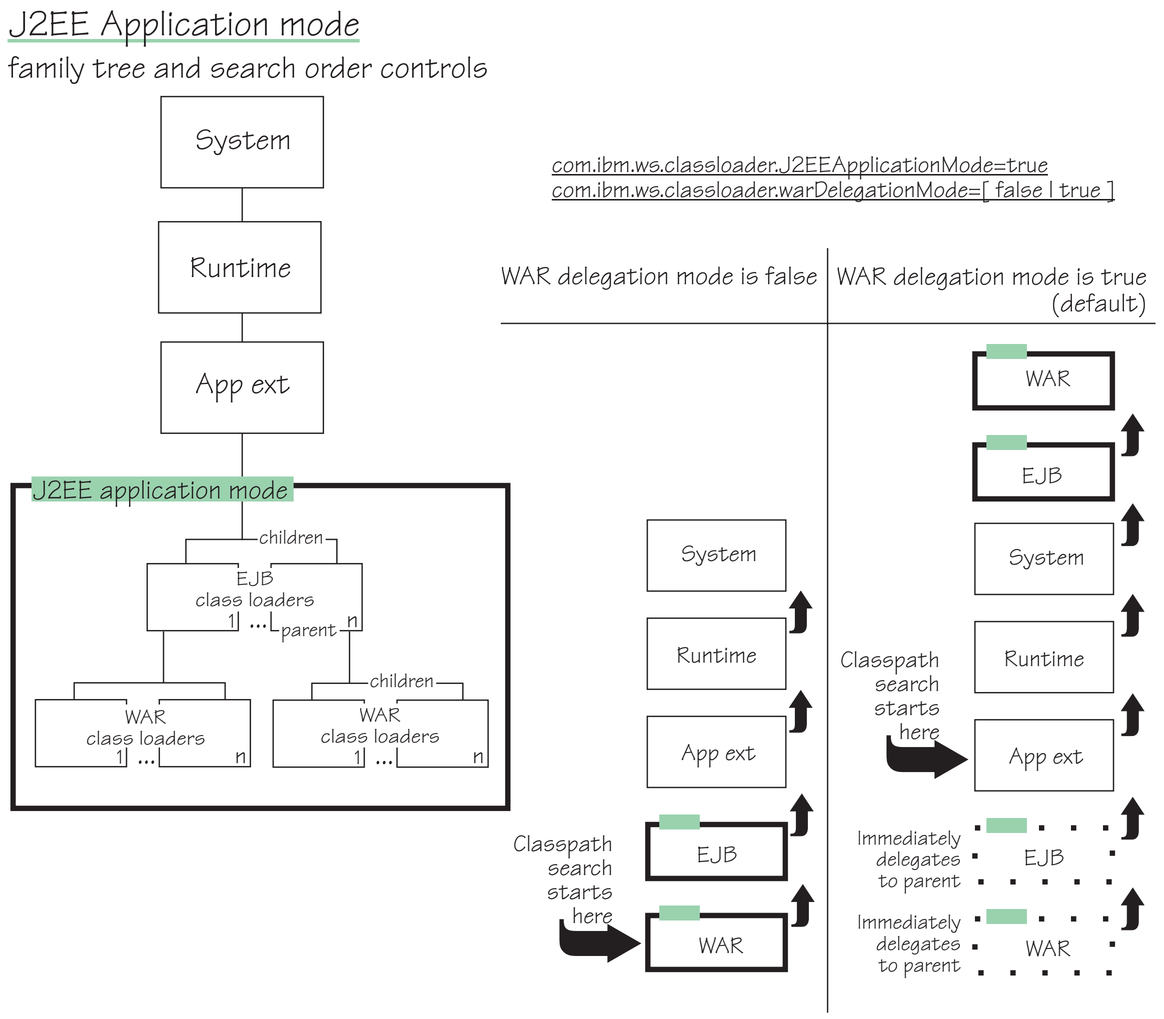

J2EE Application mode
family tree and search order controls
System
Runtime
App ext
J2EE application mode
children
EJB class loaders
1 ... n
parent
children
WAR class loaders
1 ... n
WAR class loaders
1 ... n
com.ibm.ws.classloader.J2EEApplicationMode=true
com.ibm.ws.classloader.warDelegationMode=[false | true]
WAR delegation mode is false
WAR delegation mode is true (default)
System
Runtime
App ext
EJB
WAR
Classpath search starts here
WAR
EJB
System
Runtime
App ext
Classpath search starts here
Immediately delegates to parent
EJB
Immediately delegates to parent
WAR

Figure 27. Settings for changing the search order (delegation) for J2EE Application mode

Guidelines for setting classloader mode and application packaging

Although IBM recommends using the default classloader mode (application mode), you may alter the mode if the needs of your application warrant a change in classloader behavior. Base your choice of classloader mode on the information presented in Table 20.

Table 20. Deciding which classloader mode to use

If your application:	Then use this classloader mode:	Notes
Is composed of a number of modules packaged in only one EAR file per application, and those modules within an application need to work together. In other words, each application is complete and independent of other applications.	Application	Review the hierarchy, default search order, and controls in Figure 23 on page 127

Table 20. Deciding which classloader mode to use (continued)

If your application:	Then use this classloader mode:	Notes
Is composed of a number of modules packaged in more than one EAR file, and those modules need to work together	Server	Review the hierarchy, default search order, and controls in Figure 25 on page 129
Needs to be moved from WebSphere for z/OS Standard Edition V3.5 to WebSphere for z/OS V4.0.1	Compatibility	Review the hierarchy, default search order, and controls in Figure 24 on page 128
Is composed of multiple modules, each of which may have a distinct version of some shared code segment, and thus must be kept isolated	Module	Review the hierarchy, default search order, and controls in Figure 26 on page 130 **Recommendation:** If you are currently using module mode, which is a deprecated value, IBM recommends changing to application mode.
Requires classloader behavior that is compliant with the J2EE 1.3 specification	J2EE Application	Review the hierarchy, default search order, and controls in Figure 27 on page 131

Notes on class packaging for specific classloader modes:

- For all classloader modes, parent class loaders cannot see the classes handled by child class loaders.
- If your application modules use a MANIFEST classpath, that classpath overrides the following default classloader operation:
  - **Module mode:** All module class loaders in an application are visible to each other, and all utility JAR files in the application are visible to all module class loaders in the application.
  - **Compatibility mode:** Utility JAR files are visible to all module class loaders in all active applications.
- Package EJB JAR and WAR modules that make up an application in the same EAR module.
- For module mode: If a WAR module needs to access an EJB JAR module, use a MANIFEST classpath entry in the WAR module to document the dependency.
- Common classes used by EJB JAR and WAR modules should be put in a JAR that is added to the EAR file, and referenced in the MANIFEST classpaths of the modules.

  Alternative: You can specify shared classes on the `APP_EXT_DIR` environment variable for the J2EE server.
- Class library JAR files to be used only by WAR modules should be added to the `WEB-INF/lib` directory of the WAR module.

Steps for setting trace options for classloader operation

If you encounter classloader errors on WebSphere for z/OS, you might need to alter classloader operation or repackage application components. To help you diagnose and correct errors related to class loaders and application packaging, WebSphere for z/OS provides tracing capabilities and issues error or warning messages for reporting specific conditions related to loading application classes. The following procedure tells you how to set tracing options for classloader operation.

Recommendation: Consider setting these trace options and working with your applications in a test J2EE server to determine whether the classloader mode or application packaging produce the results you expect in a production environment. Then you can confidently move your applications to a production environment that does not have tracing enabled.

Perform the following steps to set tracing options to collect data about classloader operation:

1. On z/OS or OS/390, create a trace settings file for the J2EE server in which you will install your application, and add the following trace settings:
   - `com.ibm.ws.classloader.*=all=enabled`
   - `com.ibm.ws390.csi.WS390ApplicationManager=all=enabled`
   - `com.ibm.ws390.csi.WS390ContainerManager=all=enabled`
   - `com.ibm.ws.runtime.Server=all=enabled`

   Tip: You may use a separate line for each setting, as shown above, or may specify all on a single line, separating each with a single colon (:) to distinguish each trace setting.

2. Create a new or edit an existing JVM properties file for the J2EE server. In the properties file, add the following environment variables:
   - `com.ibm.ws390.trace.settings`, which points to the location of the trace data file you just created. For the value of this variable, specify the fully qualified directory path and file name for your trace settings file.
   - `com.ibm.ws.classloader.debug`, which sets the level of tracing detail for WebSphere for z/OS to collect. For the value of this variable, specify the numeral 3.

   Rule: The JVM properties file must reside in the same HFS subdirectory as the `current.env` file for the J2EE server. See Appendix A, "Environment and JVM properties files," on page 299 information about the location of these files.

3. Use the WebSphere for z/OS Administration application to complete the following:
   a. Define a new or modify an existing J2EE server. Check the environment variable settings to make sure the `TRACEBUFFLOC` variable includes the value `SYSPRINT`.
   b. Install your applications.
   c. Start the J2EE server.

   If necessary, see the topic on defining a server configuration in "Defining the server configuration" on page 149, for more specific instructions for installing applications and starting the J2EE server.

Chapter 7. Assembling a J2EE application

Before you can install a J2EE application in a WebSphere for z/OS J2EE server, you need to prepare the application for installation. In other words, you need to package the JAR or WAR files for individual components together into an Enterprise archive (EAR) file, and ensure that all component references can be resolved. An EAR file is an archive file similar to a JAR file, with a specific directory structure and format, and has an extension of `.ear`. To create an EAR file for your application, use a WebSphere application assembly tool. "A WebSphere application assembly tool" means either the z/OS edition of the Application Assembly tool, the WebSphere Studio Application Developer tool, the WebSphere Studio Application Developer Integration Edition tool, or the Direct Deployment Tool/390fy.

During this assembly process, you not only define the individual components of a single application, but also deploy those components for the WebSphere for z/OS environment.Various WebSphere development and assembly tools generate an `application.xml` file (part of the EAR file contents), which contains the descriptive meta-information that ties together all of the JAR or WAR files that you might package in a single application. This metadata enables the J2EE server to understand the content and individual component requirements of an installed J2EE application.

The preferred method of deploying applications is to use WebSphere Studio Application Developer and Direct Deployment Tool/390fy. If you are using the following two IBM extensions you still need to use the WebSphere for z/OS Application Assembly tool:

- **SyncToOSThread:** See "Synchronizing operating system thread identity to RunAs identity" on page 31 for more information on this extension.
- **Connection Management Policy:** See "Exploiting connection management support" on page 71 for more information on this extension.

The following table shows the subtasks and associated procedures for using the Application Assembly tool to assemble a J2EE application:

Subtask	Associated procedure (See . . .)
Learn about WebSphere for z/OS classloader operation and how to it might affect how you package and deploy applications	"Preparing applications for assembly and installation" on page 120
Use the WebSphere for z/OS Application Assembly tool to assemble a new J2EE application	• "Steps for installing the Application Assembly tool" on page 136 • "Steps for assembling a new J2EE application" on page 137
Learn about the WebSphere for z/OS Direct Deployment Tool/390fy, which you can use to assemble and deploy a new J2EE application	"Direct Deployment Tool/390fy" on page 141

© Copyright IBM Corp. 2000, 2003

Steps for installing the Application Assembly tool

Before you begin:

- Download the latest copy of the Application Assembly tool, which IBM delivers through its WebSphere Application Server web site:

  `http://www.ibm.com/software/webservers/appserv/zos_os390/support.html`

  From that site, click on `Download` in the left frame to access the Application Assembly tool.
- Make sure you review the Readme file for the Application Assembly tool, so that you understand any temporary limitations or instructions that might apply for the latest code.
- If you already have a copy of the WebSphere for z/OS Application Assembly tool installed on your workstation, remove it before installing the latest version.

Perform the following steps to install the Application Assembly tool on your workstation:

1. Download or copy the code for the Application Assembly tool to a temporary directory on your workstation.

2. In the temporary directory on your workstation, locate and select the `aat`*`xxxx`*`.exe` file for the Application Assembly tool, then double-click with the left mouse button.

   Result: The InstallShield Wizard displays the setup language window.

3. Select the default setup language (English) by clicking `OK`.

   Result: After some initial preparation, the ″Welcome″ window appears on the screen.

4. From the ″Welcome″ window for the tool, click `Next`.

   Result: The license agreement window appears.

5. Accept the terms of the license agreement by selecting the appropriate radio button, and click `Next`.

   Result: The customer information window appears.

6. Accept the default user name (`IBM user`), select the radio button to install the tool only for yourself, and click `Next`.

   Result: The setup type window appears.

7. Select the radio button to install the complete tool, and click `Next`.

   Result: The ″Ready to install″ window appears.

8. Click on the `Install` button.

   Result: A progress bar indicates status until the ″InstallShield Wizard Completed″ window appears.

9. Click the `Finish` button.

   Result: The InstallShield Wizard window closes.

You should now see a subdirectory called `WebSphere for z/OS Application Assembly` under `C:\Programs\IBM`.

Steps for assembling a new J2EE application

Before you can install a new Java 2 Enterprise Edition (J2EE) application in a J2EE server, you need to prepare the application for installation. This preparation includes using the Application Assembly tool to:

1. Import J2EE application components that you created through an appropriate application development tool, and package them together in an application.
2. Define, verify, or correct attributes in the deployment descriptor for each application component or for the application itself.
3. Export your application, which causes the Application Assembly tool to create an Enterprise archive (EAR) file, which is the input format that the Administration application requires for installing applications. An EAR file packages together all of the component code (JAR and WAR files) and the deployment descriptor for the J2EE application.

The following instructions assume that this is your first time using the Application Assembly tool. The instructions guide you through the major tasks for importing and deploying a new application. To become familiar with the tool, you may use the help information for the Application Assembly tool, by selecting `Help → Contents` and view the "Quick Start" topic.

When you begin to use the tool for assembling your own components, make sure that you have used the recommended tools and specification levels for component development, which are described in:

- Chapter 5, "Setting up the application development environment," on page 109, and
- Chapter 6, "Creating new application components to be installed in a J2EE server," on page 115.

When you first start the tool, the left frame of the window contains only a folder named `Applications`. After you begin defining application names and importing J2EE application components, the left frame displays those applications and components in a hierarchical tree structure. To perform any tasks related to those components, you may select components and tasks using either one of the following methods:

- Use the left mouse button to select an application or component label in the left frame, and then use the right mouse button to display a pop-up menu of tasks.

  Tip: As an alternative, you may use the right mouse button to click the object, which automatically selects the object and displays the pop-up menu.
- Use the left mouse button to select an application or component label in the left frame, and then use the left button to click a pull-down menu label or toolbar icon.

Before you begin:

- Make sure you have the correctly updated your workstation environment and installed the Application Assembly tool, as instructed in "Steps for installing the Application Assembly tool" on page 136.

- Review the Readme file for the Application Assembly tool, so that you understand any temporary limitations or instructions that might apply for the latest code.
- Consider having a copy of the appropriate Sun Microsystems specification (or an equivalent reference), in case you need to look up bean or servlet attributes or other information. The help information for the Application Assembly tool includes this type of information, so you might not need an additional reference.

Perform the following steps to assemble a new J2EE application for installation in a J2EE server:

1. To start the Application Assembly tool, click `Start → Programs → IBM WebSphere for z/OS → Application Assembly`
   Result: The Application Assembly tool window appears, with the `Applications` folder selected.
   Tip: By default, the Application Assembly tool has a maximum Java heap size of 512M, which may not provide a large enough heap if you are assembling applications over 20M in size. If you are going to work with such large applications, consider:
   - Specifying a larger Java heap size by setting a higher value on the `AAT_MAX_HEAP_SIZE` environment variable. For further details, use the Application Assembly tool help information by selecting `Help → Environment`
   - Packing the application in several smaller EAR files instead of one EAR file.

---

2. To define a new application, click `Selected → Add`. In the right frame, enter values for the following information:
   a. Application display name, which is the name of the folder that will appear in the left frame.
   b. (Optional) Application description, which is any text to briefly describe this new application

   Rules for application display name:
   - The name can be up to 255 characters long.
   - For portability, use only the following characters:
     – Uppercase or lowercase A through Z
     – Numbers 0 through 9
     – Period (.)
     – Underscore (_)
     – Hyphen (-)
   - Do not include any nulls or slash characters in a filename.
   - Do not use doublebyte characters in filenames.
   - Filenames are case-sensitive, so `FILE1` is not the same as `file1`.

---

3. Click the diskette icon to save these values. Under the `Applications` folder, a new node appears with a label matching the application display name you just entered.
4. Expand the application node to display folders for the types of application components you may import.

---

5. To import application components, highlight the folder for the type of application component you want to import, and select `Import`. From the Import dialog:

a. Click the `Choose` button and select the JAR or WAR file you want to import into this application.
b. Click `Select` to enter the full path name for JAR or WAR file.
c. Click `OK` to start the import process.

Guidelines: Some application components might require additional files to be packaged in the EAR file. The following guidelines identify potential additions to the EAR file, with suggestions for limiting the number of application parts or avoiding duplication of files. You may import these additional files using the Import button on the Files property.

- For Web components, make sure you import the `web.inf` file.
- For CMP beans with finder helpers, include the associated `vaprt.jar` file.
- For all CMP beans, include the `ivjejb35.jar` file.
- For any components that require specific utilities or functions, include the JAR files for those functions. If application components or their z/OS or OS/390 clients share the same utilities or functions, consider excluding those JAR files from the EAR file. Instead, you can transfer those JAR files to z/OS or OS/390, and include them on the CLASSPATH variable for:
  – Any z/OS or OS/390 client that needs those functions, and
  – Any J2EE server in which the application components are installed.

---

6. Expand the view to list the components in the JAR or WAR file you just imported.

   As the application tree expands to display a component in an imported file, the Application Assembly tool displays a message if it detects errors in any deployment descriptors.

   Tip: To display more detailed information about these errors, either select `File → Message log`, or click the message log toolbar icon. Then you can use the message log as a checklist for the errors you must correct.

   Recommendation: Although you can correct errors in components, JAR, or WAR files using the Application Assembly tool, you should make corrections using the application development tool (such as WebSphere Studio Application Developer and Application Developer Integration Edition, or VisualAge for Java) that you used to create the application component.

---

7. Change or update the properties associated with each application component, using the following process. These properties are the attributes that appear in the deployment descriptor for each application component.

   Repeat this process for each component that comprises your J2EE application. If you need help for any of the component properties, click the right mouse button to select the property, and then select `Help` from the pop-up menu.

   a. Click `Selected → Modify` for an application component you want to change. The component's properties are displayed in the right frame of the Application Assembly tool window.
   b. Use the tabs in the right frame of the window to navigate through the various properties. Most of these component properties correspond to the appropriate Sun Microsystems specification; other properties are IBM extensions to the specification.
   c. After completing changes to a selected application component, save your changes. When you save your changes, the Application Assembly tool

detects any errors in the changed property values, and displays more detailed information in the message log.

Rules:

- You cannot select another component until you have either saved or cancelled changes for the currently selected component.
- You must correct any errors that the Application Assembly tool detects, or runtime results will be unpredictable.

Guidelines: If the Application Assembly tool did not detect any errors when you imported an application or component, you are not required to modify any of the application or component properties for applications to be installed in a J2EE server. You might, however, want to do the following:

- Rename applications or components to match any naming conventions your installation might recommend for applications installed on z/OS or OS/390.

  Rule: Bean names must be unique, within a given JAR file.
- Decide whether the application requires the following:
  - Container-transaction elements
  - Security roles and method permissions

    If you want to use security roles for Enterprise beans or servlets, define role names in the deployment descriptor for either individual components or for the J2EE application. Application-level roles override component-level roles. For additional instructions, see Chapter 11, "Using security roles and RunAs identities with Enterprise beans," on page 231.

---

8. Repeat steps 5 through 7 for all of the components that you want to assemble into a single J2EE application.

---

9. To validate the contents of an application, select the application in the tree, then select `Validate`.

   Tip: Validating each component individually, after finishing the steps in 5 through 7, might be faster than validating the entire application.

---

10. To deploy an application, select the application in the tree, then select `Deploy`. Message BBO94009I appears in the status bar when the deployment process is complete.

---

11. To export a deployed application, select the application in the tree, then select `Export`. The Export application window opens.

---

12. In the Export application window, you may enter the full path name for a new or existing EAR file, or click `Choose` to browse for an existing EAR file or an appropriate location for a new EAR file.

    Recommendation: When exporting your own applications, consider setting up and using a specific folder or subdirectory where you can easily find applications that are ready for installation in the J2EE server on z/OS or OS/390.

    Result: The Application Assembly tool creates new or updates existing XML files for your application, and for each of the JAR or WAR files for the components. These XML files contain the values you entered (if any) for the components' properties, and enable the J2EE server to understand the content of and correctly manage an installed J2EE application.

Samples:

- EAR file contents for an application containing two WAR files and one EJB JAR file:

```
/usr/MyApp

   EJB123.jar
   webappABC.war
   myItems.war

   /meta-inf
      application.xml
      manifest.mf
```

- `application.xml` file contents for the same EAR file:

```
<?xml version="1.0 encoding="ISO-8859-1"?>
<application>
<display-name>MyApp</display-name>

>module>
  <web>
    <web-uri>webappABC.war</web-uri>
    <context-root>/Payroll</context-root>
  </web>
</module>

<module>
  <web>
    <web-uri>myItems.war</web-uri>
    <context-root>/MyTools</context-root>
  </web>
</module>


<module>
  <ejb>EJB123.jar</ejb>
</module>

</application>
```

You know you are done when message BBO94010I appears in the status bar, indicating that your application prodfamily exported to the EAR file.

Direct Deployment Tool/390fy

Direct Deployment Tool/390fy is a command line processor that allows you to do reference and resource resolution and assign JNDI names (mapping). This functionality will allow the users who are starting to move their development environment from Visual Age Java and WebSphere Studio to a new integrated J2EE development environment tooling, WSAD (WebSphere Application Developer). This new tooling will enable a user to directly import and deploy their J2EE applications (EAR files) without requiring a trip through an application assembly tool. As WSAD starts to dominate developers' popularity in building and testing J2EE applications, the need for separate application assembly will slowly be diminishing. It is designed to close the gap between the WebSphere for Distributed and WebSphere Application Server V4.0.1 for z/OS and OS/390.

Direct Deployment Tool is also meant to provide a command line utility tool which can be used to allow advanced deployers to scriptify their deployment process without requiring a GUI based deployment tool, WebSphere for z/OS Administration application, for resolving their J2EE applications. This function will allow users to take a J2EE compliant EAR file and directly feed it into their customized scripts to resolve and deploy them onto WebSphere for z/OS and OS/390 directly. The new command line direct deployment tool, called 390fy, can be called on the input ear file to generate or replace the input ear file, and the resulting ear file can then be fed into the SM Scripting API's earfile processing call for deployment onto a selected target J2EE server.

If you use the Administration and Operations Applications to deploy your applications, the Administration and Operations Applications automatically run the 390fy program to resolve your ear files. In this situation you have no need to run the 390fy command. However, if you deploy your applications through some other method, typically through the System Management Scripting API, you must run the 390fy command to resolve your ear files for use on z/OS and OS/390.

The same 390fy command ships with both the Administration and Operations applications and the WebSphere Application Server for z/OS and OS/390 runtime. For a description of the new command line utility for direct deployment, 390fy, and instructions for using it to deploy J2EE enterprise applications, see *WebSphere Application Server V4.0.1 for z/OS and OS/390: System Management Scripting API*, SA22-7839.

Note: The preferred method of deploying applications is to use WebSphere Studio Application Developer and 390fy. If you are using the following two IBM extensions you still need to use the Application Assembly tool:

- **SyncToOSThread:** See the section "RunAs identities" on page 30 for more information on this extension.
- **Connection Management Policy:** See the following for more information on the Connection Management Policy extension:
  - "Exploiting connection management support" on page 71
  - The Application Assembly tool's built-in Help
  - The Beta Connector Guide located at:

    `http://www.ibm.com/software/webservers/appserv/download_v4z.html`

Chapter 8. Creating a J2EE server run-time environment

Now that you have created an EAR file for your J2EE application, you are ready to install and run your application in an existing or a new WebSphere for z/OS J2EE server. This chapter provides step-by-step instructions for creating a new J2EE server for your application.

As shown in Figure 2 on page 6, a J2EE server is only one component of the WebSphere for z/OS run-time, which also contains the System management server, Naming server, as well as servers in which your applications run. This complete run-time must be in place before you may create a J2EE server. Ordinarily, during the WebSphere for z/OS installation and customization process, your system programmers set up a complete run-time that includes the BBOASR2 J2EE server, which they use to run installation verification programs. You may model your new J2EE server on BBOASR2 or on another existing J2EE server.

The J2EE server, or run-time environment, consists of the following elements:
- A generic J2EE server that represents the application environment. A server is an entity that is responsible for a certain type of work.
- A J2EE server instance in which your application will run. A server instance consists of one control region, and at least one server region. The control region accepts work requests, and the server region is the actual run-time environment for the application, which includes a Java virtual machine (JVM), an EJB container, and possibly a Web container. (If you are installing a J2EE application containing servlets or JSPs, you must complete some additional tasks to configure a Web container for the J2EE server.)
- A J2EE resource and J2EE resource instance, which identify a generic type of data management subsystem and a specific data management subsystem, respectively. This resource might be, for example, the subsystem that manages a persistent data store for components installed in the J2EE server.
- A J2EE resource connection that enables the components in the J2EE server to access the resource. The Administration application automatically creates this connection when you install your J2EE application.

The following instructions for creating a J2EE server include sample names to use for these elements:
- **J2SERV** for the generic J2EE server
- **J2SERV1** for the J2EE server instance

You do not have to use the sample names; however, if you choose different names, you must follow the rules listed in “Steps for completing manual z/OS or OS/390 tasks” on page 144 to set up your application environment correctly.

The instructions also tell you where to find sample files that you may copy and modify to create z/OS or OS/390 artifacts for a J2EE server.

The following table shows the subtasks and associated procedures for creating a J2EE server for your application:

Subtask	Associated procedure (See . . .)
Completing manual z/OS or OS/390 tasks	“Steps for completing manual z/OS or OS/390 tasks” on page 144

© Copyright IBM Corp. 2000, 2003

Subtask	Associated procedure (See . . .)
Creating JCL procedures	"Steps for creating JCL procedures for the control and server regions" on page 146
Setting JVM properties	"Steps for setting properties for the JVM" on page 147
Enabling the J2EE server to host Web applications (optional)	"Steps for enabling J2EE server support for Web applications (optional)" on page 147: 1. "Steps for setting up an HTTP server" on page 368 2. "Steps for configuring the Web container" on page 158 3. "Steps for adding the J2SERV server" on page 151, if you are using the HTTP Transport Handler to handle HTTP protocol requests. 4. "Steps for configuring the V3.5 run-time provided with WebSphere for z/OS" on page 371, if you are using the HTTP Server to handle HTTP protocol requests.
Defining and activating the J2EE server through the Administration application	"Defining the server configuration" on page 149

Steps for completing manual z/OS or OS/390 tasks

Depending on your installation's conventions, many of these manual tasks might have been completed already, as part of either installing and verifying the WebSphere Application Server product itself, or setting up WebSphere Application Server test or production environments. Because documentation for these manual tasks is available already, the following procedure provides only a summary of the tasks, with references to resources with further instructions. Use this procedure as a checklist to make sure you have the correct environment set up, before you begin to define a new J2EE server for testing application components.

Perform the following steps to complete the manual z/OS or OS/390 tasks related to defining a new J2EE server:

1. Decide on naming conventions for J2EE application components, J2EE server elements, and z/OS or OS/390 subsystems, such as DB2. The following instructions for creating a J2EE server include sample names to use for these elements, but you should replace them with names that your installation either has set up or prefers to use.

   For recommendations for naming conventions, see the appendix in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Operations and Administration*, SA22-7835.

2. Define the workload manager (WLM) application environment, service class, and classification rules for the new J2EE server and the applications it will host. To define the application environment (that is, to define the J2EE server to WLM), use the IWMARIN0 dialog to fill in the following values:

Field in IWMARIN0 dialog:	Value to use:
Run-time server	Use a short description of the J2EE server, such as `EJB-DB2 application server`
Application environment name	`J2SERV`
Subsystem type	`CB`

Field in IWMARIN0 dialog:	Value to use:
Procedure name	`J2SERV1`
Start parameter	`IWMSSNM=&IWMSSNM`
Limit on starting server address space for a subsystem instance	`No limit`

For further details about using the IWMARIN0 dialog, and defining service classes and classification rules, see *z/OS MVS Planning: Workload Management*, SA22-7602.

3. Set up the database resources or connectors for data access. Your system programmer or database administrator has probably installed and configured the required z/OS or OS/390 subsystems, such as DB2, and might already have created the databases or tables that your application will use. So the only tasks you might need to do are these:
   - Find out what DB2 subsystem name you need to specify when you define the J2EE server.
   - Create any database tables that your J2EE application components need to use.

   If necessary, see *DB2 Administration Guide*, SC26-9931 for instructions on creating database tables.

4. Define security profiles and permissions, using your installation's security product. You might need to work with your installation's security administrator to accomplish this task. The security profiles and permissions depend, to some degree, on your installation's guidelines for test or production systems. For example, in a test environment, you might allow J2EE application clients to access test systems and data without using any security mechanism. This approach might be especially suitable when client programs run on the same z/OS or OS/390 system as the J2EE server.

   Guidelines:
   - Regardless of the security you set up for client access to resources, certain authorizations are required for the J2EE server. For example, if your J2EE application requires the use of DB2, the J2EE server needs to be granted access to the DB2 plan `DSNJDBC`. For recommendations and instructions for setting up security for J2EE servers and J2EE application clients, see *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*.
   - To enable WebSphere for z/OS to delete temporary, application-created files when an application is uninstalled from a J2EE server, you need to define **both** of the following to the same RACF group:
     – The server region identity that you specify when you create the J2EE server; this identity is the user ID under which the server region runs. This user ID must match an entry in the RACF STARTED class and have appropriate RACF authorizations for a server region.
     – Any systems management user IDs (for example, CBADMIN) that might be used to start the WebSphere for z/OS Administration application to uninstall applications from this J2EE server.

You also must change the default umask value that sets permission bits for application-created files, as instructed in "Steps for creating JCL procedures for the control and server regions."

For further information about RACF authorizations for server regions, systems management user IDs, and file permissions, see the section on setting up security in Chapter 2 of *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*.

- If you want to install a J2EE application that requires role-based security, you need to define profiles in the EJBROLE or GEJBROLE class, and then allow users or groups to have read access to those profiles.

  Rules:

  – Profiles specified in the EJBROLE or GEJBROLE class follow this format:

    ```
    role_name
    ```

    where *role_name* matches the security role attribute specified in either:
    - The J2EE application deployment descriptor, or
    - The deployment descriptor of an individual application component.

  – A role name cannot contain blanks, and cannot exceed 246 characters. Role names, however, may be in mixed case.

  If your installation uses the z/OS or OS/390 SecureWay Security Server (RACF), see *z/OS Security Server RACF Command Language Reference*, SA22-7687 for information about using:

  – The RDEFINE command to define profiles to the EJBROLE or GEJBROLE class.

  – The PERMIT command to grant users read access to these profiles.

Steps for creating JCL procedures for the control and server regions

In TSO, perform the following steps to set environment variables and create JCL procedures for the application control region and server region:

1. In your working PROCLIB data set, create a new member named J2SERV (the generic server name). Copy the BBOASR2 sample member from BBO.SBBOJCL into this new member, and make appropriate updates according to comments in the file. Modify the PROC statement to use the **server instance** name you will specify in the WebSphere for z/OS Administration application. For example, the PROC statement should state something like this:

   ```
   //J2SERV    PROC SRVNAME='J2SERV1'
   ```

2. Also in your PROCLIB, create a new member named J2SERV1 (the JCL procedure name you will later specify to WLM). Copy the BBOASR2S sample member from BBO.SBBOJCL into this new member, and make appropriate updates according to comments in the file. For example:

   - Edit the IWMSSNM parameter to use the server instance name you will specify in the WebSphere for z/OS Administration application:
     `IWMSSNM='J2SERV1'`
   - | Change the default umask value so that the user IDs for the server region and systems management user IDs (such as CBADMIN) have write permission to remove temporary, application-created files when the application is uninstalled from the J2EE server. On the JCL EXEC statement, specify the following:

```
PARM='ENVAR("_EDC_UMASK_DFLT=00x")
")
```
where 00x is the umask value to use. The default value is 002.
Recommendation: A umask value of 002 will cause files to be created with permission bits set to 775. This is the IBM recommended value.
For further information about RACF authorizations for server regions, systems management user IDs, and file permissions, see the section on setting up security in Chapter 2 of *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*.

Steps for setting properties for the JVM

Use the following procedure only if you want to change the default settings that WebSphere Application Server uses for the Java virtual machine (JVM) that runs in the J2EE server. To change the defaults, create a JVM properties file, specifying the properties and values that you want to use.

Before you begin:

- Review the supported JVM properties listed in "JVM properties and properties files" on page 339, which also contains information about the placement and content of a JVM properties file.
- You might need special authorization to edit an existing JVM properties file, or store a new file in the appropriate directory. Check with the system programmer who installed WebSphere Application Server on your test system.

Perform the following steps to set up a JVM properties file:

1. Edit an existing or create your own JVM properties file, and place it in the same HFS directory in which WebSphere Application Server places the current.env file containing environment variables for this J2EE server.

   Rule: This JVM properties file must be named `jvm.properties`

2. Edit the JVM properties file to add or set the property keys and values that you want to use.

3. Save your changes to the JVM properties file.

If WebSphere Application Server cannot find or use the property file you provide, it continues the process of activating the server, using default JVM property values.

Steps for enabling J2EE server support for Web applications (optional)

If you are installing a J2EE application that contains only Enterprise beans, you do not have to perform any of the steps in this section. In this case, skip to "Defining the server configuration" on page 149. If your application contains servlets or JSPs, however, you must complete some additional tasks to set up the J2EE server configuration.

Before you begin: You need to:

- Read "The WebSphere for z/OS environment for Web applications" on page 80, which introduces terms and concepts that you need to understand before completing any of the related subtasks and procedures.

- Decide which servlet execution environment you want to use, and how to receive and direct inbound servlet requests. The following table lists the options:

Table 21. Deciding how to configure the environment for Web applications

If your Web applications will run in this execution environment:	**Then you need to configure the following to receive inbound requests:**	**Notes on configuration instructions:**
The WebSphere for z/OS J2EE server	One or more of the following: • A WebSphere for z/OS HTTP Transport Handler[1] • An IBM HTTP Server and the V3.5 runtime provided with the V4.0.1 product	• To set up the WebSphere for z/OS HTTP Transport Handler, follow these instructions: 1. Define the `BBOC_HTTP_`*`xxx`* environment variables as part of completing the instructions in"Steps for adding the J2SERV server" on page 151, and 2. Follow the instructions in "Steps for configuring the Web container" on page 158 • To set up the IBM HTTP Server and the V3.5 runtime, follow these instructions: 1. "Steps for setting up an HTTP server" on page 368 2. "Steps for configuring the V3.5 run-time provided with WebSphere for z/OS" on page 371 3. "Steps for configuring the Web container" on page 158
The V3.5 runtime in an HTTP Server address space	The IBM HTTP Server	Follow these instructions: 1. "Steps for setting up an HTTP server" on page 368 2. "Steps for configuring the V3.5 run-time provided with WebSphere for z/OS" on page 371 3. "Steps for configuring the Web container" on page 158

1. The HTTP Transport Handler currently does not support the following:
   - The authentication policy specified in your Web application's deployment descriptor. As an alternative, your security administrator can define a surrogate user ID under which all requests received will execute.
   - HTTPS. This limitation means that your installation cannot use SSL connections for inbound servlet requests.

- If you are running Web applications in both the V3.5 runtime and the J2EE server, decide whether you must use the default `was.conf` configuration file provided with the V4.0.1 product to configure the V3.5 runtime. You can copy properties you want to continue using (such as webapp and deployedwebapp properties) from your Standard Edition V3.5 `was.conf` configuration file to this default `was.conf` file.

  The default `was.conf` file already includes the `webapp` and `deployedwebapp` properties for the sample Web application used for installation verification. If you use the default file without changing it, any requests for your own Web applications (i.e., Web applications for which there are no `webapp` and `deployedwebapp` properties) will be routed to the J2EE server. If these Web applications are being hosted by the V3.5 runtime instead of the Web container, you will receive an error message that the application could not be found.

  If you want to run some of your applications Standard Edition V3.5 Web applitionsin the V3.5 runtime, you must copy the `webapp` and `deployedwebapp`

properties for these applications from your existing Standard Edition V3.5`was.conf` file into the V4.0.1 default `was.conf` file.
- Decide which names to use for Web container virtual hosts and context root definitions. This decision requires knowledge of the domain names through which inbound servlet requests will arrive, and knowledge of individual Web application modules. For background information about virtual hosts and context roots, see “Resolving requests to a specific servlet using the HTTP Server” on page 374.

The following table shows the subtasks and associated procedures for enabling support for Web applications installed in a WebSphere Application Server J2EE server:

Subtask:	**Associated procedure (See ...)**
Setting up the HTTP Transport Handler to establish communication between the J2EE server and a Web browser	“Steps for adding the J2SERV server” on page 151
Configuring a Web container in the J2EE server to run servlets and JSPs	“Steps for configuring the Web container” on page 158
Setting up an HTTP server to establish communication between the J2EE server and a Web browser	1. “Steps for setting up an HTTP server” on page 368 2. “Steps for configuring the V3.5 run-time provided with WebSphere for z/OS” on page 371

Defining the server configuration

Use the WebSphere Application Server Administration application, also known as the System Management End-User Interface (SM EUI), to define the run-time environment for your application. Defining this run-time environment, or server configuration includes defining a J2EE server, server instance, datasource; and installing the EAR file for your J2EE application.

Recommendation: Define the environment variable settings at the server level. When you do so, these settings apply for all server instances.

Before you begin: You should know:
- Where to find additional help with using the Administration application: Help is available in the Administration application itself, and in *WebSphere Application Server V4.0.1 for z/OS and OS/390: System Management User Interface*, SA22-7838.
- Which environment variables that you need to set for the run-time environment. See Appendix A, “Environment and JVM properties files,” on page 299 for a complete list of run-time variables and the values you can set during this process.

The following table shows the subtasks and associated procedures for defining a J2EE server configuration, using the WebSphere Application Server Administration application:

Subtask:	**Associated procedure (See ...)**
Starting the Administration application	“Steps for starting the Administration application” on page 150
Starting a new conversation	“Steps for starting a conversation” on page 151

Subtask:	Associated procedure (See ...)
Adding the server	"Steps for adding the J2SERV server" on page 151
Adding the server instance	"Steps for adding the J2SERV1 server instance" on page 153
Adding the J2EE resource	"Steps for adding a J2EE resource" on page 153
Adding the J2EE resource instance	"Steps for adding the J2EE resource instance" on page 153
Installing the J2EE application	"Steps for installing a J2EE application" on page 154
Validating the new conversation	"Steps for validating the new conversation model" on page 156
Committing the conversation	"Steps for committing the conversation" on page 157
Marking manual tasks as completed	"Steps for marking z/OS or OS/390 tasks as completed" on page 157
Activating the new conversation	"Steps for activating the server configuration" on page 157

Steps for starting the Administration application

When you login to the WebSphere for z/OS Administration application, you will be able to view only your own conversations and the current active conversation, even if you have the same security authorities of other administrator user IDs.

Recommendation: If your installation has multiple administrators, they are able to login to the Administration application using the same user ID, but should not login simultaneously.

Before you begin:

- You need to know the naming server IP name and port number for the machine running WebSphere Application Server. The naming server IP name is set either in your domain name server (DNS) or workstation `HOSTS` file; the default port number is 900.
- You might need to increase the heap size associated with the Administration application, if you want to avoid delays when installing very large Enterprise Archive (EAR) files. To increase the heap size on Windows 2000, for example, complete the following steps:
  1. Select `Start → Programs → IBM WebSphere for z/OS`
  2. Right-click on `Administration` and select `Properties`
  3. On the `Shortcut` tab, specify the following for the `Target` field:

     ```
     "D:\Program Files\Ibm\WebSphere Application Server for zOS and OS390\
        bin\smdrv.exe" "-J-mx128m"
     ```

Perform these steps to start the Administration application:

1. On your workstation, `Start → Programs → IBM WebSphere for z/OS → Administration`
2. Fill in the dialog with the naming server IP name, port `900`, the user ID `CBADMIN`, and password `cbadmin`. Click OK. Wait for the message that indicates initialization is complete.

You know you are done when the main dialog window appears.

Steps for starting a conversation

Perform these steps to start a new conversation:

1. Select the Conversations folder with the left mouse button. Then, using the right mouse button, click on the Conversations folder, then select Add.

   ____

2. In the properties form (the panel on the right), enter a name for the new conversation.

   ____

3. Click on the save (diskette) icon. The words ″Adding... Conversation″ appear in the tree.

   ____

You know you are done when message BBON0515I appears in the status bar (at the bottom of the dialog window), indicating that the new conversation was added.

Steps for adding the J2SERV server

Perform these steps to add the new server:

1. Expand your new conversation tree by clicking on the node to the left of the conversation name.

   ____

2. Expand Sysplexes, then your sysplex.

   ____

3. Select the J2EE server folder with the left mouse button. Then, using the right mouse button, click on the J2EE server folder, then select Add.

   ____

4. In the properties form, enter values or make selections as appropriate for your installation.

   Usually, you can use default values for most of the properties; however, make sure you check at least the properties listed in the following table. For a complete list and explanation of server properties, use the help available through the Administration application, or see *WebSphere Application Server V4.0.1 for z/OS and OS/390: System Management User Interface*, SA22-7838.

Server name	J2SERV
Control region identity	The user ID under which the control region runs. This user ID must match an entity in the RACF STARTED class and have appropriate RACF authorizations for a control region.
Server region identity	The user ID under which the server region runs. This user ID must match an entity in the RACF STARTED class and have appropriate RACF authorizations for a server region.
Local identity	Use only if you want to allow non-authenticated clients
Remote identity	Use only if you want to allow non-authenticated clients
Control region start procedure name	J2SERV

Notes:

a. Also in the properties form, provide values for the following key environment variables for the application server. Make sure you set all **required** environment variables for the run-time environment. See Appendix A, “Environment and JVM properties files,” on page 299 for a complete list of application server run-time variables and their values. For more information about how to add or modify environment variables, use the help system in the Administration application or see *WebSphere Application Server V4.0.1 for z/OS and OS/390: System Management User Interface*, SA22-7838.

- LIBPATH. The LIBPATH variable specifies the DLL search paths for Java and JDBC in the hierarchical file system (HFS). Specify system, WebSphere Application Server, Java, and DB2 JDBC DLLs.

  Example:

  ```
  LIBPATH=/db2_install_path/lib
  :/usr/lpp/java/IBM/J1.3/bin
  :/usr/lpp/java/IBM/J1.3/bin/classic
  :/usr/lpp/WebSphere/
  ```

  where *db2_install_path* is the HFS where you installed DB2 Universal Database for z/OS and OS/390.
- CLASSPATH. The CLASSPATH statement specifies Java class files (JAR files and classes.zip) for use by Java applications in server regions.

  If the CLASSPATH variable does not already contain a value, copy the value set for the sysplex in this conversation, and append any necessary files. If your application components access DB2 data, add the full path to the zip file for the JDBC driver.

  Rule: The entire CLASSPATH contents must fit on **one** line.

  Example:

  ```
  CLASSPATH=:/usr/lpp/ldap/lib/ibmjndi.jar
  :/db2_install_path/classes/db2j2classes.zip
  ```

  Notes:

  1) Application classes should not be manually added to this classpath. If you have added application classes to this classpath, they should be removed and specified on the APP_EXT_DIR environment variable. Leaving application classes on the system classpath could have unpredictable results.
  2) After activation of this conversation, WebSphere Application Server automatically prepends the following files to the J2EE server CLASSPATH for you:
     - `ws390srt.jar`
     - `waswebc.jar`
     - `xerces.jar`

b. (Optional) Set up the WebSphere for z/OS HTTP or HTTPS Transport Handler to receive inbound servlet requests for Web applications installed in this J2EE server. See“Setting up the HTTP/HTTPS Transport Handler” on page 83 for a description of how to set up the HTTP and HTTPS Transport Handlers.

---

5. Click on the save (diskette) icon. The words "Adding... J2EE server" appear in the tree.

You know you are done when message BBON0515I appears in the status bar, indicating that the new server definition was added.

Steps for adding the J2SERV1 server instance

Perform these steps to add the server instance:

1. If necessary, expand the J2SERV folder by clicking on the node to the left of the folder icon.
2. Select Server Instances with the left mouse button. Then, using the right mouse button, click on Server Instances, then select Add.
3. In the properties form, enter J2SERV1 as the server instance name.
4. Optionally, enter a server instance description.
5. Optionally, supply a log stream name. If you do not supply one, the default is the log stream name you chose for the J2SERV server.
6. Click on the save (diskette) icon. The words ″Adding... Server Instance″ appear in the tree.

You know you are done when message BBON0515I appears in the status bar, indicating that the new server instance was added.

Steps for adding a J2EE resource

Perform these steps to add a J2EE resource:

1. Select J2EE Resource with the left mouse button. Then, using the right mouse button, select Add.
2. In the properties form, enter a name for the J2EE resource.
3. Optionally, enter a description of the J2EE resource.
4. Find the property labelled `Datasource type`, and select `DB2`.

   The Administration application fills in the fields above with the information that is appropriate for a DB2 datasource.
5. Click on the save (diskette) icon. The words ″Adding... J2EE resource″ appear in the tree.

You know you are done when message BBON0515I appears in the status bar, indicating that the J2EE resource was added.

Steps for adding the J2EE resource instance

Perform these steps to add the datasource instance:

1. If necessary, expand the tree for the newly created J2EE resource by clicking on the node to the left of the resource name.

2. Select J2EE resource instance with the left mouse button. Then, using the right mouse button, click on J2EE resource instance, then select Add.

---

3. In the properties form, enter the appropriate values for the following:
   - J2EE resource instance name
   - J2EE resource instance description
   - Location name (for example: supply the DB2 Universal Database for z/OS and OS/390 location name)

     Note: If you are using DB2 as a backing datasource, you may also supply an SQL ID to control the use of unqualified table references in your application components. For additional information about SQL IDs, see “Overview of SQLID for managed datasources” on page 34.

---

4. Click on the save (diskette) icon. The words ″Adding... J2EE resource instance″ appear in the tree.

You know you are done when message BBON0515I appears in the status bar, indicating that the J2EE resource instance was added.

Steps for installing a J2EE application

Before you begin: Make sure that the ftp server on z/OS or OS/390 is running.

Note: The container implicitly assigns the NotSupported transaction attribute to each container-managed transaction bean method to which no transaction attribute has been assigned. If you do not want the container to assign the NotSupported transaction attribute, transaction attributes can be assigned in an application assembly tool prior to installing the J2EE application.

Perform the following steps to install the EAR file for your application, using the Administration application:

1. In the tree, select the J2EE server in which you want to install your application.

---

2. Choose Install J2EE Application... from the Selected menu bar. The Install J2EE Application dialog box appears.

---

3. In the dialog box, enter the following values:
   - The fully qualified path name of the EAR file that contains your J2EE application.
   - The name of the FTP server for the sysplex in which you want to install your application. Usually, this is the server IP name you specified as instructed in “Steps for starting the Administration application” on page 150.
   - Click OK.

     Result: The Reference and Resource Resolution window appears, and displays the application content in the EAR file.

---

4. For each Enterprise bean listed in the Reference and Resource Resolution window, click on the bean name to display the details for that bean on the right side of the window.

Complete the following steps, as necessary, for each bean. Tab names on the right side of the window indicate whether you need to complete specific steps; tab names with a checkmark do not require any actions on your part.

a. Click on the EJB tab, and then click on the Set Default JNDI Name button.

b. Click on the Reference tab to list any beans that this bean references. Under the label JNDI Name, click on the ↓ symbol to display a list of possible JNDI names for each referenced bean, and select the appropriate JNDI name.

   Repeat this step for each bean in the Reference list.

   Note: The Link column contains checkboxes with checkmarks for all those bean references that have been statically bound to another bean within the application, using the `ejb-link` element. If you used the Application Assembly tool to generate such links, you cannot change them now. You can, however, generate additional links by resolving a reference to a bean that resides within the application you are currently installing.

c. Click on the Resource tab to display the datasources for this bean. Under the label Datasource, click on the ↓ symbol to display a list of possible JNDI names for the datasource, and select the appropriate JNDI name. Usually, this name is `db2os390:`*`ssn`*, where *ssn* is the DB2 Universal Database for z/OS and OS/390 subsystem name that you specified when adding the datasource instance.

   If this bean is an entity bean that uses container-managed persistence (CMP), the `ws390rt/cmp/jdbc/CMPDS` resource reference appears under the Resource tab for this CMP bean. This resource reference was added to your application's deployment descriptor during assembly of the application, to allow you to select the datasource that WebSphere Application Server will use to back CMP beans.

   Rule: When you install the application, you must select a datasource to back any CMP beans.

Tip: Data from the Reference and Resource Resolution window is saved in a new copy of the EAR file named *`application_name`*`_resolved.ear` before it is transferred to the server for deployment. If you reopen that copy of the file later, you do not have to re-enter the information a second time.

---

5. Repeat the JNDI selection process for any remaining beans. You will know you have finished this process for each bean, when the bean symbol to the left of the bean name has a checkmark over it.

---

6. For each servlet listed in the Reference and Resource Resolution window, click on the servlet name to display the details for that servlet on the right side of the window. Then complete the following steps, as necessary, for each servlet:

   a. Set default JNDI names any referenced components.

   b. Set JNDI names for any J2EE resources that the servlet requires.

---

7. Repeat the JNDI selection process for any remaining servlets. You will know you have finished this process for each servlet, when the symbol to the left of the servlet name has a checkmark over it.

---

8. When the JNDI selection process is complete for all application components, the OK button becomes selectable. Click OK.

   Result: This action starts the automatic ftp transfer of the EAR file contents from your workstation to z/OS or OS/390. The message Deploying... *application_name* appears on the screen. The ftp transfer proceeds through the following stages:

Stage	Description
1	When the ear file is imported, the system transfers it to `targetdir/sysplex/temp/administrator_ID/application_name.ear` *targetdir* is the mount point, *sysplex* is the name of the sysplex, and *administrator_ID* is the user ID of the administrator (usually CBADMIN).
2	The ear file is copied to `targetdir/apps/J2SERV/Ln/application_name.ear` *n* is the level number.
3	The ear file is processed. During ear file processing, the ear file is exploded into directory `targetdir/apps/J2SERV/Ln/app_name/` *app_name* is the name of the application (not necessarily equal to the ear file name).
4	A scaffolding directory `targetdir/apps/J2SERV/Ln/A/` is created under which all the deployment information is stored.

Note: Upon activation of the conversation, everything beneath

```
targetdir/apps/J2SERV/Ln/
```

is moved one level up to

```
targetdir/apps/J2SERV/
```

Also during this deployment process for your application:

- Appropriate ownership and file permissions are set for your application files.
- If the application contains any servlets or JSPs, these Web applications are provided with a fully qualified URI that enables the WAR files and the EJB JAR files to be accessed through HTTP protocol when requested by a client. (See "Steps for configuring the V3.5 run-time provided with WebSphere for z/OS" on page 371 for more information on invoking a Web application from a browser.)

You know you are done when message BBON0470I appears in the status bar, indicating that the `application_name_resolved.ear` file has been successfully installed.

Steps for validating the new conversation model

Perform these steps to validate the conversation:

1. If necessary, scroll up the tree to the conversation you have defined.

2. Select the conversation with the left mouse button. Then, using the right mouse button, click on the conversation, then select Validate.

You know you are done when message BBON0442I appears in the status bar, indicating that the new conversation is valid.

Steps for committing the conversation

Perform these steps to commit the conversation:

1. If necessary, scroll up the tree to the conversation you have validated.

---

2. Select the conversation with the left mouse button. Then, using the right mouse button, click on the conversation, then select Commit. Answer Yes to the question: ″Do you still want to commit?″ The words ″Committing... *conversation_name*″ appear in the tree.

You know you are done when message BBON0444I appears in the status bar, indicating that the new conversation and J2EE server definition was committed.

Steps for marking z/OS or OS/390 tasks as completed

1. Select the new conversation with the left mouse button. Then, using the right mouse button, click on the conversation, then select Instructions.

---

2. Double-check the instructions provided by the Administration application to determine whether you have completed all of the required z/OS or OS/390 tasks, which include defining workloads and setting up security. A checklist for these required z/OS or OS/390 tasks appears in “Steps for completing manual z/OS or OS/390 tasks” on page 144.

---

3. When you have verified or finished the z/OS or OS/390 tasks, mark all tasks complete in the administration application by following these steps:
   a. Select the conversation with the left mouse button. Then, with the right mouse button, click on the conversation, select Complete, then All tasks.
   b. Answer Yes to the question: ″Are you sure that all tasks have been completed?″

You know you are done when message BBON0484I appears in the status bar, indicating that all tasks are complete.

Steps for activating the server configuration

1. Select the conversation with the left mouse button. Then, with the right mouse button, click on the conversation, then select Activate.

---

2. Answer Yes to the question: ″Do you want to activate conversation *conversation_name*?″ At the bottom of the dialog, a message indicates when the server definition has been activated.

You know you are done when message BBON0449I appears in the status bar, indicating that the new conversation was activated. Now the J2EE server you just activated is ready to host the applications you installed.

Steps for configuring the Web container

The WebSphere Application Server J2EE server is a servlet execution environment. Servlets run in the J2EE server's Web container, and use the RMI/IIOP protocol to access Enterprise beans in the J2EE server's EJB container. Configuring the Web container is optional; it is required only if you are enabling J2EE server support for Web applications (see "Steps for enabling J2EE server support for Web applications (optional)" on page 147), and plan to or have already installed Web applications in the J2EE server.

A Web container is created as part of the J2EE server set up process. It's configuration settings are specified in a webcontainer.conf file provided with the product. You can update the webcontainer.conf file to:

- Configure one or more virtual hosts within a Web container. Virtual hosting allows a single Web container to handle processing for more than one internet host. For example, the same Web container may service requests for hosts **www.mycompany.com** and **www.MyOtherCompany.com**.

  You can deploy one or more Web applications into a virtual host. This capability allows the Web container configuration to be partitioned according to the hosts for which it is servicing requests. The Web container uses the **host.** properties to determine to which virtual host an application request is to be routed. It checks the URL used to initiate an input request and routes the request to the specified virtual host.
- Specify whether or not you want to collect session data. If you want to collect session data, you can also specify other settings, such as the name of the DB2 table that will be used to store session data. "HTTP session support" on page 87 provides more information about collecting session data and the options that can be set in the webcontainer.conf file.

  Note: This database table can be shared between V3.5 and V4 WebSphere Application Servers.

After editing the webcontainer.conf file, you must refresh the J2EE server to activate the changes you made.

Perform the following steps to configure the J2EE server's Web container:

1. Create a `webcontainer.conf` file by copying the default file shipped in `/usr/lpp/WebSphere/bin` directory.

   ---

2. Edit the `webcontainer.conf` file and update the following properties to define the virtual host and context roots. When the Web container is initially configured, at least one virtual host (the default virtual host that is provided with the product) is already associated with it. The following properties are used to configure a virtual host:
   - **host.**<*virtual-hostname*>**.alias**=<*hostname*>

     Use this property to specify the hostname alias to be associated with this virtual host name. It provides a binding between the host names understood by the HTTP Transport Handler or the HTTP Server, and the virtual host definitions in the Web container. The alias can be the name by which this virtual host is known to clients and applications.

- **host.***<virtual-hostname>***.mimetypefile**

  Use this property to specify 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 virtual host. If this property is not specified, the Web container uses the standard MIME type definitions provided in the default_mimetype.properties file.
- **host.***<virtual-hostname>***.contextroots**

  Use this property to bind installed Web applications into a specific virtual host. The specified context root corresponds to the context root assigned to the Web application during application deployment. The Web container's default configuration includes a predefined virtual host, named **default_host**, and a contextroot property that binds all installed Web applications to the **default_host** virtual host.

  If you are defining only one virtual host per J2EE server, you can use the default context root binding property. All subsequently installed applications will be bound to this virtual host.

See Appendix B, "Default webcontainer.conf file," on page 351 for more a more detailed descriptions of these properties.

Guidelines:

- The easiest way to configure the Web container is to use a single virtual host and a universal context root; these definitions are probably sufficient if you are testing Web applications, and familiarizing yourself with using the Web container as the servlet execution environment. For this simple configuration, you can use the following definitions:

  ```
  host.default_host.alias=host_name_of_HTTP_server
  ⋮
  host.default_host.contextroots=/
  ```
  With this context roots definition, the single forward slash provides a universal catch-all for applications being bound to the default virtual host. Every application, regardless of the context root definition in the `application.xml` file, will bind to this virtual host.
  Rule: Do not define the same context root (which includes the universal / value) for more than one virtual host definition. Results will be unpredictable.
- When you want to have multiple host names routed to the same z/OS or OS/390 system, see:
  - "Resolving requests to a specific servlet using the HTTP Server" on page 374 for an overview of using virtual hosts and context roots, and
  - Appendix B, "Default webcontainer.conf file," on page 351 for more details about each of the properties you can use in the webcontainer.conf file.

Note: A virtual host can have more than one alias, and each alias definition may contain both a host name and a port number. All of the host's alias names must be specified on a single line separated by a comma and a space.

See Appendix B, "Default webcontainer.conf file," on page 351 for a complete description of the webcontainer.conf properties that are applicable to defining a virtual host.

3. Edit an existing or create a new Java virtual machine (JVM) properties file for the J2EE server in which your Web applications will be installed. This properties file include the following property, which identifies the location of the Web container configuration file, usually named `webcontainer.conf`:

   ```
   com.ibm.ws390.wc.config.filename=/path/your_webcontainer.conf_filename
   ```

   If this property is not added to the JVM properties file, the Web container uses the default file, *applicationserver_root*/bin/webcontainer.conf.

   Note: Even though the file system location of the webcontainer.conf file is optional, you might want to place the webcontainer.conf file in the same directory as the other configuration files associated with this J2EE server.

   "JVM properties and properties files" on page 339 contains further instructions for creating this properties file, including location and access requirements.

   Tips:

   - Make sure you double-check the property name, path, and file names you specify. Any misspellings or mistakes will cause the J2EE server to assume your file does not exist, and the J2EE server will use default JVM values instead. These default values do not identify a Web container configuration file. Without a Web container configuration file, the WebSphere for z/OS plug-in will not be able to verify any inbound servlet requests as valid.
   - Do not try to copy an existing `was.conf` file and edit it for use as a `webcontainer.conf` file. At first glance, these files might look similar, but they are not. You will not save any time or effort trying to reuse a `was.conf` file. Instead, you may use the model in Appendix B, "Default webcontainer.conf file," on page 351.

4. Refresh the J2EE server to pick up the changes to the `webcontainer.conf` file, and any changes to the optional JVM properties file.

Steps for configuring HTTP Session Support

Before you begin: Configure an HTTP(S) Transport Handler to handle HTTP(S) requests to the Web container, or have an HTTP Server set up on your sysplex to perform this function. (If you are using DB2 Session Persistence Version 2 to maintain your session data in a DB2 database, you must use an HTTP(S) Transport Handler.)

You need to know that the WebSphere for z/OS Web container contains a single Session Manager. The Session Manager supports the javax.servlet.http.HttpSession interface described in the Java Servlet API 2.2 specification. It also supports two versions of DB2 session persistence. The Session Manager is initially configured such that:

- Sessions are enabled.
- Cookies, for conveying session IDs to servlets, are enabled.
- Session objects are stored in-memory rather than in a DB2 database.
- Cookies can be exchanged in both HTTPS and HTTP sessions.
- The ability to add session IDs to URLs in transition from HTTP to HTTPS and back (protocol switch rewriting) is disabled.
- URL rewriting, for conveying session IDs to servlets, is disabled.

The following sections describe how to change these session settings to better fit your installation's requirements.

You can also update the other session properties in the webcontainer.conf file to change such session settings as the size of the session table used to maintain session objects within the Web container, and the amount of time in, milliseconds, that a session is allowed to go unused before it is no longer considered valid. Detailed descriptions of all of the session properties are provided in the copy of the default webcontainer.conf file that is provide in Appendix B. Default webcontainer.conf file of *Assembling Java™2 Platform, Enterprise Edition (J2EE™) Applications*.

Note: If you make any changes to the session configuration settings, you must stop and start the affected J2EE server instance again before any of these changes take affect.

Configuring cookies

If cookies are to be used with session tracking, the following changes might need to be made to one or more of the following webcontainer.conf file properties:

1. Set the `session.cookies.enable` property to `true` to enable cookies.

   ---

2. (Optional) Specify the name of the cookie on the `session.cookie.name` property if you can not use the default cookie name JSESSIONID.

   Notes:

   a. IBM recommends that you **DO NOT** change the default cookie name (JSESSIONID) unless it is absolutely necessary to do so.
   b. If you are using a non-z/OS WebSphere plug-in for Web servers that is at a V4.0.2, V4.0.3, or V4.0.4 level to provide session affinity across multiple J2EE server instances, you **MUST NOT** change the default cookie name (JSESSIONID). These plug-ins, by default, look for affinity data in a cookie named JSESSIONID.
   c. If you are using the WebSphere HTTP Plug-in for z/OS or a non-z/OS WebSphere plug-in for Web servers that is at a V4.0.5 level or above to provide session affinity across multiple J2EE server instances, you can change the cookie name, for a J2EE server but if you do, you **MUST** also:
      1) Use the WebSphere for z/OS Administration application to add the `SESSION_COOKIE_NAME` environment variable to the current.env file. The value specified for this variable must exactly match the value specified on the `session.cookie.name` property in the webcontainer.conf file. (This value is case sensitive.)
      2) Edit the plugin-cfg.xml file for the Web server plug-in, and add the `affinityCookie="`*new_cookie_name*`"` parameter to all of the `<Uri name>` elements contained within the <UriGroup>tag for that J2EE server. The same value **MUST** be specified for *new_cookie_name* for all of the URIs contained in this URI group.

         Example: If the cookie name, MyCookie, was specified on the `SESSION_COOKIE_NAME` environment variable to the current.env file, and on the `session.cookie.name` property in the webcontainer.conf file, you must specify this same value on the `affinityCookie` parameter for each of the `<Uri name>` elements contained within the `<UriGroup>` tag for that J2EE server:

```
<UriGroup name="Uris">
      <Uri name = "/servlet/snoop" affinityCookie="mycookie"/>
      <Uri name = "/webapp/*" affinityCookie="mycookie"/>
      <Uri name = "*.jsp" affinityCookie="mycookie"/>
</UriGroup>
```

If the value specified for the `SESSION_COOKIE_NAME` environment variable, the `session.cookie.name` webcontainer.conf file property, and the <CloneID=> element in the plugin-cfg.xml file do not all match exactly, session affinity will not work for this J2EE server instance.

---

3. Set the `session.cookie.maxage` property to a specific time interval. This change is only needed if you want the cookie to persist for a set length of time instead of for the full duration of the invocation of a browser. (The value specified must be an integer value that indicates, in milliseconds, the amount of time the cookie is to remain valid.)

---

4. Set the `session.cookie.domain` property to a specific name if you want to limit the domain for which a cookie is valid.

---

5. Add a `session.cookie.comment` property if you want to include a comment about the cookie.

---

6. Set the `session.cookie.secure` property to `true` if you want to restrict the exchange of cookies to only HTTPS sessions.

Configuring URL rewriting

If you need to use URL rewriting instead of cookies, you must make the following changes to webcontainer.conf file properties:

1. Set the `session.urlrewriting.enable` to `true` to enable URL rewriting in the Session Manager.

---

2. Set the `session.protocolswitchrewriting.enable` property to `true` to enable the session ID to be added to a URL when the URL requires a switch from HTTP to HTTPS, or HTTPS to HTTP.

---

3. Set the `session.cookies.enable` property to `false` to disable the use of cookies as a way to manage sessions. (If both cookies and URL rewriting are enabled, the Session Manager will attempt to use cookies to manage sessions and ignore the fact that URL rewriting is enabled.

Notes:

1. If you have APAR PQ67436 installed, you can use URL rewriting even if you are:
   - Maintaining session data in memory across multiple server regions within a single J2EE server instance.
   - Maintaining session data in memory across multiple J2EE server instances, provided you are using one of the following WebSphere plug-ins for Web servers:
     - The WebSphere HTTP Plug-in for z/OS that is used with the Version 5.3 IBM HTTP Server for z/OS and OS/390. (This plug-in is provided in WebSphere for z/OS Service Level W401500.) This plug-in

(ihs390WASPlugin_http.so) should not be confused with the Local Redirector plug-in (was400plugin.so) that is shipped with the WebSphere for z/OS product to provide an IIOP connection from an IBM HTTP Server for z/OS and OS/390 to a WebSphere for z/OS Web container.

  – One of the WebSphere plug-ins for Web servers that run on a non-z/OS Web server that is shipped with WebSphere for z/OS and with the WebSphere Application Server Advanced Edition Version 4.0.5 or higher product. See Table 22 on page 176 for a list of these plug-ins.

2. If you need 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.

Configuring WebSphere for z/OS to maintain session data in a DB2 database instead of in-memory.

WebSphere for z/OS can be configured to maintain session data in a DB2 database instead of in-memory. Using a DB2 database, session data can be maintained within the same or across multiple WebSphere for z/OS server regions or J2EE server instances. Therefore, maintaining session data in a DB2 database provides session failover support since another server region/server instance can take over in the event of a catastrophic failure in the initial server region/server instance.

There are two versions of DB2 session persistence that can be used with WebSphere for z/OS:

- Session Persistence Version 1 is the version that was shipped with versions 3.5 and 4.0, and initially shipped with version 4.0.1. You should continue to use Version 1 if:
  1. Your installation needs to share the data stored in the DB2 database amongst WebSphere for z/OS instances and Version 3.5 Application Servers executing on a z/OS or OS/390 image that are able to access this central database, or
  2. Your installation must use the WebSphere for z/OS local redirector plug-in and a Web server to handle HTTP(S) requests.
- Session Persistence Version 2 was added to Version 4.0.1 with PTFs UQ90051 and UQ90052. It provides improved performance and added functionality over Version 1, and should be used if Version 1 is not required for either of the reasons specified in the previous bullets.

To set up DB2 Session Persistence Version 1:

1. Use the WebSphere for z/OS Administration application to modify the instance definition of the IBMHttpSession J2EE Resource. Make sure you include the DB2 Location Name in the instance definition. The resulting datasource definition is used to get connections to the DB2 database containing the session table.

   ---

2. Have your DB2 Administrator create a DB2 database table for storing session data. (For more information about creating DB2 databases see the *DB2 Administration Guide* for the version of DB2 you are using.)

   The table space in which the database table will be 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;
```

The DB2 table defined within this table space, that the Session Manager will use to process the session data must have the following format:

```
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_name>.<tablespace_name>;
```

Note: The length attributes specified for VARCHAR in this example are not necessarily the values your DB2 Administrator should use for the DB2 table he is creating. See the DB2 SQL Reference for the version of DB2 you will be using for guidance in determining appropriate values for these length attributes for your installation.

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_name>.<index_name>
  ON <database_name>.<table_name>
  (ID , PROPID)
  USING STOGROUP <group_name>
   ERASE NO
  BUFFERPOOL BP0
  CLOSE YES;
```

Notes:

a. At run time, the Session Manager will access the target table using the identity of the J2EE server in which the owning Web application is deployed. Any Web container that is configured to use persistent sessions should be granted both read and update access to the subject database table.

b. HTTP session processing uses the index defined using the CREATE INDEX statement to avoid database deadlocks. In some situations, such as when 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 recommendations on how to calculate the space required for an index, and adjust the size of the tables you are using accordingly.

c. It may be necessary to tune DB2 in order to make efficient use of the sessions database table and to avoid deadlocks when accessing it. Your DB2

Administrator should refer to the DB2 Administration Guide for specific information about tuning the version of DB2 you are using.

---

3. Make sure that the following property settings are specified in the webcontainer.conf file to enable DB2 Session Persistence Version 1 and to specify how you want the Session Manager handle the maintenance of the session data. If the `session.persistenceversion` property is not included in the webcontainer.conf file, DB2 Session Persistence Version 1 will be used, since 1 is the default value for this property.

   ```
   session.enable=true
   session.invalidationtime=<milliseconds>
   session.dbenable=true
   session.persistenceversion=1
   session.dbtablename=<database_name.table_name>
   ```
   <*milliseconds*> is the amount of time in, milliseconds, that a session is allowed to go unused before it is considered invalid.
   <*database_name.table_name*> is the name of the database and DB2 table that is to be used by session services.

   Note: The HTTP session timeout settings specified in the webcontainer.conf file can be overridden for a particular Web application by adding the following tags to the the web.xml file for that application:

   ```
   <session-config>
        <session-timeout>x</session-timeout>
   <session-config>
   ```

   where x is the timeout value, in minutes, for that application. You can also override the webcontainer.conf file setting by specifying the new value in the ″Session timeout″ field on the ″General″ tab for the Web application when you use the WebSphere for z/OS AAT to deploy the Web application on your WebSphere for z/OS system.

   In addition, one of the following properties needs to be specified:

   ```
   session.cookies.enable=true
   ```
   or
   ```
   session.urlrewriting.enable=true
   ```

To set up DB2 Session Persistence Version 2:

1. Use the WebSphere for z/OS Administration application to modify the instance definition of the IBMHttpSession J2EE Resource. Make sure you include the DB2 Location Name in the instance definition. The resulting datasource definition is used to get connections to the DB2 database containing the session table.

---

2. Have your DB2 Administrator create a DB2 database table for storing session data. (For more information about creating DB2 databases see the *DB2 Administration Guide* for the version of DB2 you are using.)

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

```
CREATE DATABASE <database_name>
  STOGROUP SYSDEFLT
  CCSID EBCDIC;

CREATE TABLESPACE <tablespace_name> IN <database_name>
  USING STOGROUP <group_name>
  PRIQTY 512
  SECQTY 1024
  LOCKSIZE ROW
  BUFFERPOOL BP32K;
```

The Session Manager will use the DB2 table defined within this table space to process the session data. This table must have the following format:

```
CREATE TABLE <database_name>.<table_name>  (
  ID               VARCHAR(95) NOT NULL ,
  PROPID           VARCHAR(95) NOT NULL ,
  APPNAME          VARCHAR(64) ,
  LISTENERCNT      SMALLINT ,
  LASTACCESS       DECIMAL(19,0),
  CREATIONTIME     DECIMAL(19,0),
  MAXINACTIVETIME  INTEGER ,
  USERNAME         VARCHAR(256) ,
  SMALL            VARCHAR(3122)  FOR BIT DATA ,
  MEDIUM           VARCHAR(28869) FOR BIT DATA ,
  LARGE            BLOB(2097152),
  SESSROW          ROWID NOT NULL GENERATED ALWAYS
  )
  IN <database_name>.<tablespace_name>;
```

Note: The length attributes specified for VARCHAR in this example are not necessarily the values your DB2 Administrator should use for the DB2 table he is creating. See the DB2 SQL Reference for the version of DB2 you will be using for guidance in determining appropriate values for these length attributes for your installation.

A unique index must be created on the ID and PROPID columns of this table. The following is an example of the index definition:

```
CREATE UNIQUE INDEX <database_name>.<index_name>.
     <database_name>.<table_name>
      (ID      ASC,
       PROPID  ASC,
       APPNAME ASC);
```

Notes:

a. At run time, the Session Manager will access the target table using the identity of the J2EE server in which the owning Web application is deployed. Any Web container that is configured to use persistent sessions should be granted both read and update access to the subject database table.

b. HTTP session processing uses the index defined using the CREATE INDEX statement to avoid database deadlocks. In some situations, such as when 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 recommendations on how to calculate the space required for an index, and adjust the size of the tables you are using accordingly.

c. It may be necessary to tune DB2 in order to make efficient use of the sessions database table and to avoid deadlocks when accessing it. Your DB2 Administrator should refer to the DB2 Administration Guide for specific information about tuning the version of DB2 you are using.

A large object (LOB) table space must be defined and an auxiliary table must be defined within that table space. The following is an example of the LOB table space definition:

```
CREATE LOB TABLESPACE <LOB_tablespace_name> IN <database_name>
    BUFFERPOOL BP32K
    USING STOGROUP <group_name>
    PRIQTY 512
    SECQTY 1024
    LOCKSIZE LOB;

CREATE AUX TABLE <database_name>.<aux_table_name>
    IN <database_name>.<LOB_tablespace_name>
    STORES <database_name>.<table_name>
    COLUMN LARGE;
```

An index must be created for this auxiliary table. The following is an example of the index definition:

```
CREATE INDEX <database_name>.<aux_index_name> ON
      <database_name>.<aux_table_name>;
```

Finally, your DB2 Administrator must grant the appropriate server regions access to these DB2 tables. The following is an example of the command used to grant server region CBASRU1 access to the tables you just created:

```
GRANT ALL ON <database_name>.<table_name> TO CBASRU1;
```

3. Make the following changes to the webcontainer.conf file session properties:
   - Set the `session.dbenable` property in the webcontainer.conf file to `true` to enable DB2, and the `session.persistenceversion` property to 2 to enable the use of the new DB2 table format (required). If the `session.persistenceversion` property is not included in the webcontainer.conf file, WebSphere for z/OS will attempt to use DB2 Session Persistence Version 1, since 1 is the default value for this property. This will cause session-related errors to occur during application processing.
   - Specify one of the following properties (required):

     ```
     session.cookies.enable=true
     ```
     or
     ```
     session.urlrewriting.enable=true
     ```
   - Change the following webcontainer.conf file properties, if necessary, to fit your installation's requirements (optional):
     - Set the `session.reaperinterval` property to the time interval, in seconds, at which you want the reaper (i.e., invalidator) to run. If this property is not specified, or if the specified value is less than 30, WebSphere for z/OS will calculate an appropriate interval.
     - Set the `session.timebasedwrite` property to true if you want updates to the DB2 database to be done on a separate thread, and set the `session.timebasedwriteinterval` property to the time interval, in seconds, at which you want this thread to write session updates to the DB2 database. (If no value is specified for the `session.timebasedwriteinterval` property, the default value of 120 seconds is used.) Setting this property to true should result in improved performance but leaves a larger window for data loss if a failure occurs. (This property can be left at its default value of false if retention of data is more important than performance.)
     - Set the `session.dbconnections` property to the number of DB2 database connections you want the Session Manager to hold for its exclusive use. Holding connections will improve performance. By default, each JVM runs three threads, in which case, setting this property to 3 DB2 connections is

optimal. If no value is specified for this property, no connections will be held for the Session Manager's exclusive use.

– Set the `session.usingmultirow` property to true, if you want each session attribute written in a separate row, or to false, if you want all session attributes written in a single row. If your session is used only for a few small attributes, specifying false will most likely result in better performance. If no value is specified, the default value of true is used.
– Set the `session.writeallproperties` property to true if you want to write all properties to the DB2 database even if they have not been changed by a call to setAttribute. If you only want the properties written if they have been changed, leave this property set to the default value of false, which should result in improved performance.
– Set the `session.scheduledinvalidation` property to true if you want to invalidate sessions at two specific hours of the day and then use the `session.scheduledhour1` and `session.scheduledhour2` properties to specify these two hours. This minimizes database access for invalidations, and effectively means that sessions will not timeout when the time limit specified on the `session.MaxInactiveInterval` property is reached. Specifying true should result in improved performance.

  Note: If you use this property, IBM recommends setting the `session.reaperinterval` property to 3600 seconds (1 hour). This will enable session invalidation to occur sometime during the hours specified on the `session.scheduledhour1` and `session.scheduledhour2` properties. (This invalidation will not necessarily occur right on the hour.)

Note: Values specified for these properties will be ignored if the `session.persistenceversion` property is set to 1 or is not included in the webcontainer.conf file.

---

4. If you want to maintain session data in an environment with multiple server instances, either:
   - Install a Version 5.3 IBM HTTP Server on a z/OS or OS/390 system and use FTP or another file transfer mechanism to download the WebSphere HTTP Plug-in for z/OS from your WebSphere for z/OS system to that HTTP Server, or
   - Install a WebSphere plug-in for Web servers on a supported non-z/OS Web server running on a distributed platform workstation.

   Then configure the Web server and the plug-in to communicate with the appropriate WebSphere for z/OS J2EE server instances. (See "Configuring session affinity across multiple WebSphere for z/OS J2EE server instances" on page 169 for a description of how to complete this step.)

   Note: Session affinity **CAN NOT** be maintained across multiple WebSphere for z/OS J2EE server instances if the TCPIP port on which the HTTP(S) Transport Handler is listening is configured as a SHAREPORT.

Configuring session data to be stored in-memory

HTTP session support is initially configured to maintain session data in-memory. If you changed these initial settings in order to maintain session data in a DB2 database, to return to the initial settings make the following updates to the webcontainer.conf file:

- Set the `session.dbenable` property to `false`.

- Make sure session is set up to use either cookies (see "Configuring cookies" on page 161) or URL rewriting (see "Configuring URL rewriting" on page 162).

- Set the `session.tableoverflowenable` property to `true` if you want HTTP session data to continue to be stored as long as there is memory available. Setting this property to true can potentially exhaust system memory and even allow for system sabotage. Somebody could write a malicious program that continually hits your site and creates HTTP sessions, but ignores any cookies or encoded URLs and never utilizes the same HTTP session from one HTTP request to the next.

  When overflow is not allowed (the property is set to false), the Session Manager will still return an HTTP session with the HttpServletRequest's getSession(true) method if the memory limit has currently been reached, but the HTTP session will not be saved. Therefore, a new session will have to initiated each time a request is made until enough memory becomes available to begin saving session data again.

  If you are concerned about HTTP sessions not being saved, include the isOverflow() method of the WebSphere extension to HttpSession, com.ibm.websphere.servlet.session.IBMSession, in your application to check for this situation and react accordingly.

- Set the `session.tablesize` to the number of in-memory HTTP sessions that you want to be maintained. (The default number is 1000 session objects.) Once this number is surpassed, HTTP session data is no longer stored unless the session.tableoverflowenable property is set to true.

  General memory requirements for your hardware system, as well as your site's usage characteristics, should be considered before changing the default value specified for this property. If you specify a larger number, you may need to increase the heap sizes of the Java processes for WebSphere for z/OS instances.

Configuring session affinity across multiple WebSphere for z/OS J2EE server instances

If you are maintaining session data in a DB2 database and using DB2 Session Persistence Version 1, session affinity is not necessary. However, if you are maintaining session data in-memory, or are using DB2 Session Persistence Version 2, you must perform the following steps to provide session affinity support across multiple WebSphere for z/OS J2EE server instances.

Note: Session affinity **CAN NOT** be maintained across multiple WebSphere for z/OS J2EE server instances if the TCPIP port on which the HTTP(S) Transport Handler is listening is configured as a SHAREPORT.

1. Either:
   - Install a Version 5.3 IBM HTTP Server for z/OS or OS/390 and the WebSphere HTTP Plug-in for z/OS on the same z/OS or OS/390 on which you are running WebSphere for z/OS, or

- Install a non-z/OS WebSphere plug-in for Web servers on a supported non-z/OS Web server running on a distributed platform workstation.

  Then configure the Web server and Web server plug-in to communicate with an HTTP(S) Transport Handler on your WebSphere for z/OS system. See "Steps for setting up WebSphere plug-ins for Web servers for use with WebSphere for z/OS" on page 171 for a description of how to install and configure supported Web servers and Web server plug-ins.

  You must stop and start the Web server again before any configuration changes you make will take affect.

---

2. For each WebSphere for z/OS J2EE server instance to which requests will be redirected, make the following changes to the webcontainer.conf file:
   - Add the hostname and port where your Web server is running to the `host.<virtual_hostname>.alias` property. For example, if your Web server is running on affinity.raleigh.ibm.com port 80, you would add `www.mycompany.com:8220`, and `www.mycompany.com` to the list of aliases included on this property.
   - Make sure session is set up to use either cookies ("Configuring cookies" on page 161) or URL rewriting ("Configuring URL rewriting" on page 162).

Configuring session data sharing within a J2EE application

If you have an J2EE application that requires session data to be shared cross all of the Web modules in that application, you must perform the following steps to activate this support in a WebSphere for z/OS environment:

1. Create a file called sessionshare.xml file that contains the following tags:

   ```
   <sessionsharing>
        <EnterpriseAppnames> Appname1,Appname2,Appname3,...
        </EnterpriseAppnames>
   </sessionsharing>
   ```

   The names of the J2EE applications requiring the sharing of session data within their Web modules are listed on the <EnterpriseAppnames> tag name, separated by commas. Session data cannot be shared across J2EE applications.

   Note: This file must be in ASCII format.

---

2. Place the sessionshare.xml file in a directory that is readable by WebSphere for z/OS. (The file must also be readable by the identity under which the J2EE server is running.)

---

3. Use the Administration application to add the path for that directory to the CLASSPATH setting for this J2EE server or server instance. For example, if you added the sessionshare.xml file to the directory /u/websphere/data/, you would add that directory path to the CLASSPATH setting.

   Note: You **MUST** include the trailing slash when you specify the directory path.

---

4. If you are using a DB2 database to maintain your session data, ensure WebSphere for z/OS is configured to use DB2 Session Persistence Version 2.

(See "Configuring WebSphere for z/OS to maintain session data in a DB2 database instead of in-memory." on page 163 for a description of how to configure DB2 Session Persistence Version 2.)

5. Restart the application server. The Session Manager will read the sessionshare.xml file by default and enable the session sharing feature for the J2EE applications listed in this file.

Steps for setting up WebSphere plug-ins for Web servers for use with WebSphere for z/OS

WebSphere plug-ins for Web servers are shipped as part of the WebSphere for z/OS product. However, only the WebSphere HTTP Plug-in for z/OS, which is used with the IBM HTTP Server for z/OS and OS/390 can be run on a z/OS system. The other Web server plug-ins must be run on a non-z/OS system. You must download these non-z/OS plug-ins to a Web server that is already installed on a workstation and then configure the plug-in for that Web server. Use Table 22 on page 176 to determine the correct plug-in for your platform.

Once you have WebSphere for z/OS and your Web server and plug-in properly configured, you can route requests from your browser, through the Web server and plug-in, to one of the WebSphere for z/OS J2EE server instances defined in the ServerGroup element in the plugin-cfg.xml file. New requests will get sprayed across these server instances, but once a session is established, requests will get routed back to the correct HTTP(S) Transport Handler based on the CloneID the WebSphere for z/OS Web container assigned to the original request.

For example, a WebSphere for z/OS Web container might assign a CloneID of A to a request that was received on port 8083 and a CloneID of B to a request that was received on port 8084. During the same session, the Web server plug-in will then use these CloneIDs to redirect future requests back to the correct J2EE server instance. The HTTP(S) Transport Handler then routes the requests to the correct server region following normal processing procedures.

Setting up the WebSphere HTTP Plug-in for z/OS

To enable WebSphere for z/OS to use theWebSphere HTTP Plug-in for z/OS with an IBM HTTP Server for z/OS and OS/390, perform the following steps:

1. Make sure a Version 5.3 IBM HTTP Server for z/OS and OS/390 is installed on a z/OS or OS/390 system.
2. Add ServerInit, Service, and ServerTerm directives to the Web server's httpd.conf configuration file:
   - The following ServerInit and ServerTerm directives to indicate the entry points to the plugin's initialization and exit routines. These routines exist as entry points init_exit, and term_exit, respectively, within the ihs390WASPlugin_http.so DLL.

     ```
     ServerInit /usr/lpp/WebSphere/WebServerPlugIn/bin
        ihs390WASPlugin_http.so:init_exit <fully_qualified_path>
     ServerTerm /usr/lpp/WebSphere/WebServerPlugIn/bin/
        ihs390WASPlugin_http.so:term_exit
     ```

     where <fully_qualified_path> is the fully qualified path to the plugin-cfg.xml file.
   - The following Service directive for each application that will be using the WebSphere HTTP Plug-in for z/OS. This directive indicates the entry point

to the plug-in's request routine. The request routine exists as the entry point service_exit within the ihs390WASPlugin_http.so DLL.

```
Service /<webapp_contextroot>/* /usr/lpp/WebSphere/WebServerPlugIn/bin/
   ihs390WASPlugin_http.so:service_exit
```

where <webapp_contextroot> is the context root of the Web application.

Notes:

a. In this discussion, the ServerInit and Service directives are split for printing purposes. In the actual httpd.conf file, each of these directives should be entered on a single line.

b. 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.

c. The ihs390WASPlugin_http.so DLL is found in the **/usr/lpp/WebSphere/WebServerPlugIn/bin** directory.

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, if APAR PQ58541is applied to your HTTP Server, add the following directive to the httpd.conf file:

```
ServiceSync On
```

3. If you want to use SSL, configure the HTTP Server for SSL and set the SSLClientAuth directive to PASSTHRU.

   The WebSphere HTTP Plug-in for z/OS' SSL connection to the J2EE server uses the SSL session established by the HTTP Server. (See *z/OS HTTP Server Planning, Installing, and Using, Version 5.3 IBM* , SC34–4826, or *OS/390 HTTP Server Planning, Installing, and Using, Version 5.3 IBM* , SC31–8690, for a description of how to configure the HTTP Server for SSL.)

   Setting the SSLClientAuth directive to PASSTHRU enables the HTTP Server to request certificates from clients but keeps the HTTP Server from doing any validity checking. The HTTP Transport Handler will perform the validity checking when it receives the certificate.

4. (Optional) If you want incoming requests to be selectively authorized and refused, set the WEB_SECURITY_VERSION property in the jvm.properties file to 2.

5. Use the WebSphere for z/OS Administration application to add the following environment variables to the current.env file of each WebSphere for z/OS J2EE server instance to which requests will be redirected:

   - `BBOC_HTTP_PORT` and/or `BBOC_HTTP_SSL_PORT`

     These environment variables must specify the WebSphere for z/OS ports to which the WebSphere HTTP Plug-in for z/OS should redirect these requests. These same ports must be specified on a <Transport Hostname> element in the HTTP Plug-in for z/OS' plugin-cfg.xml file.

   - `BBOC_HTTP_MODE=INTERNAL` and/or `BBOC_HTTP_SSL_MODE=INTERNAL`

     These environment variables enable the HTTP(S) Transport Handler to trust private headers received from the HTTP Server's plug-in, over the port specified on the `BBOC_HTTP_PORT` and/or `BBOC_HTTP_SSL_PORT` environment variables.

Note: If you add these environment variables to the current.env file, the HTTP(S) Transport Handler will trust all private headers it receives. Therefore, you must ensure that there are no untrusted paths to the HTTP(S) Transport Handler.

6. Add a **host.default_host.alias** property to the webcontainer.conf file that specifies the hostname and port number for the hosting V5.3 HTTP Server. This value **must** match the value specified for this Web server on the <Virtual Hostname> element in the plugin-cfg.xml file for the WebSphere HTTP Plug-in for z/OS.
7. If your V5.3 HTTP Server is located on a different z/OS system than your WebSphere Application Server:
   a. Use FTP or another file transfer mechanism, to download, in binary format, the WebSphere Plug-in for z/OS from the WebSphere for z/OS **/usr/lpp/WebSphere/WebServerPlugIn/bin** directory to the hosting V5.3 HTTP Server.
   b. Using an authorized z/OS user ID, issue the following commands from an OMVS command line prompt to turn on the ″p″ bit in the HFS where the WebSphere HTTP Plug-in for z/OS is now located:
   ```
   chmod 777 ihs390WASPlugin_http.so
   extattr +p ihs390WASPlugin_http.so
   ```
8. Configure the plug-in. The WebSphere HTTP Plug-in for z/OS is configured using a plugin-cfg.xml file. This file **MUST** be in EBCDIC format.

   Following is a copy of the plugin-cfg.xml file template that is provided with the WebSphere for z/OS product.

   This template contains the following configuration specifications:
   - LogLevel, which can be set to ″Warn″, ″Error″, or ″Trace″. ″Trace″ should only be used for debugging purposes.
   - ServerGroup, which is a name by which you can group multiple servers.

     For each server within a ServerGroup, you can specify a CloneID attribute. By default, the Web container assigns a CloneID of <ServerName.ServerInstanceName>.

     If you prefer a different CloneID, you can use the WebSphere for z/OS Administration application to set the CLONEID environment variable in the current.env file. The value specified on this environment variable overrides the default value.

     If you specify a value on the CLONEID environment variable, you must also change the value specified for a Server element in the Web server plug-in's plugin-cfg.xml file to match this value.
   - VirtualHostGroup provides a logical grouping of a set of host definitions. The virtual host name specified in this set of tags must match the host name and port on which the HTTP server is listening for HTTP/HTTPS requests.
   - UriGroup provides a logical grouping of a set of URIs.
   - Route links the ServerGroup, UriGroup, and VirtualHostGroup definitions together.

   Notes:
   a. If you already have a plugin-cfg.xml file you are using with a different plug-in, you can make a copy to use with your WebSphere HTTP Plug-in for z/OS. If you use such a copy, you should be aware that:

1) The values specified for such elements as **ServerGroup Name** and **VirtualHostGroup** might need to be updated to reflect your WebSphere for z/OS environment.
2) The Property name=keyring and the Property name=stashfile elements will be ignored if they are left in the plugin-cfg.xml file. The WebSphere HTTP Plug-in for z/OS uses the SSL setup specified in the HTTP server's httpd.conf file and does not look for these elements in the plugin-cfg.xml file.

b. If you want to use a third party load balancer in between the plug-in and the application server, add a ClusterAddress attribute to each ServerCluster element in the plugin-cfg.xml file. The ClusterAddress attribute must specify the IP address of the load balancer you want to use. If this attribute is not included or if a value of 0 is specified, the plug-in will handle load balancing.
c. Additional elements and attributes that can be included in a plugin-cfg.xml file are described in "Properties of WebSphere plug-ins for Web servers" on page 183.

Note:

The template is located in the following directory:

```
/usr/lpp/WebSphere/WebServerPlugIn/bin/
```

```
<?xml version="1.0"?>
<Config>
   <!-- The LogLevel controls the amount of information that gets written to
        the plugin log file. Possible values are Error, Warn, and Trace.
        It also specifies the location of the log file. The time the
        information was written to the log file and the process ID will be
        appended to the file name.-->
   <Log LogLevel="Trace" Name="/directory/plugin.log">

   <!-- Server groups provide a mechanism of grouping servers together. -->
   <ServerGroup Name="my_servers_group">
      <Server Name="myserver_server1">
         <!-- The transport defines the hostname and port value that the
              Web server plug-in will use to communicate with the application
              server. In the following example, the IBM HTTP Server for z/OS
           is listening on port 8220 and the internal transport for the
              application server is listening on port 9080.-->
         <Transport Hostname="www.mycompany-1.com" Port="9080" Protocol="http"/>
      </Server>
   </ServerGroup>

   <!-- Virtual host groups provide a mechanism of grouping virtual
          hosts together. -->
   <VirtualHostGroup Name="my_server_vhosts">
      <VirtualHost Name="www.mycompany.com:8220"/>
   </VirtualHostGroup>

   <!-- URI groups provide a mechanism of grouping URIs together. Only
        the context root of a web application needs to be specified unless
        you want to restrict the request URIs that get passed to the application
        server.  -->

   <UriGroup Name="my_application_URIs">
      <Uri Name="/myapp/*"/>
   </UriGroup>

   <!-- A route ties together each of the above components. "/myapp/*" is
          the context root of the application "myapp" installed in the
          application server.-->
```

```
    <Route ServerGroup="my_servers_group" UriGroup="my_application_URIs"
        VirtualHostGroup="my_server_vhosts"/>

</Config>
```

For more information about creating your plugin-cfg.xml file, see the instructions provided in the section ″Properties of WebSphere plug-ins for Web servers″ in the WebSphere Application Server Advanced Edition InfoCenter. An online version of this InfoCenter is available at URL:

`http://www.ibm.com/software/webservers/appserv/doc/v40/ae/infocenter/index.html`

9. Stop WebSphere for z/OS and the HTTP Server and and start them again.

   The configuration is complete. In order to activate the configuration, stop and restart both the Web server and the application server. If the Web server plug-in for Version 5.3 of the IBM HTTP Server for z/OS and OS/390 sucessfully initializes when the HTTP Server is started again, you will receive the following message:

```
EJS3090I  WebSphere HTTP Plug-in for z/OS and OS/390 Version 4.00 Service
          Level 1 is starting
EJS3091I  WebSphere HTTP Plug-in for z/OS and OS/390 initializing with
          configuration file : /u/ws4.0/ga/plugin/conf/plugin-cfg.xml
EJS3093I  WebSphere HTTP Plug-in for z/OS and OS/390 initialization went OK
:-)
```

Setting up the Web server plug-in for a non-z/OS Web server

To enable WebSphere for z/OS to use a Web server plug-in, perform the following steps:

1. Make sure a supported Web server is installed on your workstation. (See Table 13 on page 93 for a list of supported Web servers for each platform.) If you need to install a Web server, follow the instructions provided with that Web server.

   Note: If you are using WebSphere Application Server Advanced Edition V4.0.2 or higher for Distributed Platforms on your workstation, you can use the IBM HTTP Server that is provided with that product, along with the Web server plug-in that is appropriate for that Web server.
2. Using Table 13 on page 93, determine which plug-in to download based on the platform you are running on and the Web server that is installed on that platform.
3. Using FTP or another file transfer mechanism, download, in binary format, the appropriate Web server plug-in from the WebSphere for z/OS /usr/lpp/WebSphere/DownloadPlugins directory.

   Note: If you are using WebSphere Application Server Advanced Edition V4.0.2 or higher for Distributed Platforms on your workstation, the appropriate Web sever plug-in is automatically installed as part of the product installation process. However, if you are using a Microsoft™ Internet Information Server, see “Installing a Web server plug-in on a Microsoft Internet Information Server (IIS)” on page 183 for additional installation information.

   At least two files must be downloaded for each Web server plug-in, :
   - The plugin-cfg.xml containing the default plug-in configuration file common to all of the WebSphere plug-ins for Web servers.
   - The plug-in executable for a specific platform and Web server. This file can be found in the directory indicated in the following table.

Directory /usr/lpp/WebSphere/DownloadPlugins/plugin-cfg.xml contains the default plug-in configuration files, commonly used among all of the Web server plug-ins. These files are equivalent to the Web server plug-in files shipped with V4.0.2 of the WebSphere Application Server Advanced Edition for Distributed Platforms product.

Table 22. Location of WebSphere plug-ins for Web servers executables

Operating system	Web server	Plug-in executable file
Windows 2000/NT	IBM HTTP Server (IHS)	/usr/lpp/WebSphere/DownloadPlugins/Win32/IHS/mod_ibm_app_server_http.dll_bin
	Lotus Domino	/usr/lpp/WebSphere/DownloadPlugins/Win32/Domino/libdomino5_http.dll_bin
	Apache	/usr/lpp/WebSphere/DownloadPlugins/Win32/Apache/mod_app_server_http.dll_bin
	iPlanet (Netscape)	/usr/lpp/WebSphere/DownloadPlugins/Win32/iPlanet/libns41_http.dll_bin
	Microsoft Internet Information Server (IIS)	/usr/lpp/WebSphere/DownloadPlugins/Win32/IIS/iisWASPlugin _http.dll_bin
IBM AIX	IBM HTTP Server (IHS)	/usr/lpp/WebSphere/DownloadPlugins/AIX/IHS/mod_ibm_app_server_http.so_bin
	Lotus Domino	/usr/lpp/WebSphere/DownloadPlugins/AIX/Domino/libdomino5_http.a_bin
	Apache	/usr/lpp/WebSphere/DownloadPlugins/AIX/Apache/mod_app_server_http.so_bin
	iPlanet (Netscape)	/usr/lpp/WebSphere/Download lugins/AIX/iPlanet/libns41_http.so_bin
HPUX	IBM HTTP Server (IHS)	/usr/lpp/WebSphere/DownloadPlugins/HPUX/IHS/mod_ibm_app_server_http.sl_bin
	Lotus Domino	/usr/lpp/WebSphere/DownloadPlugins/HPUX/Domino/libdomino5 _ http.sl_bin
	Apache	/usr/lpp/WebSphere/DownloadPlugins/HPUX/Apache/mod_app_server_http.sl_bin

Table 22. Location of WebSphere plug-ins for Web servers executables (continued)

	iPlanet (Netscape)	/usr/lpp/WebSphere/DownloadPlugins/HPUX/iPlanet/libns41_http.sl_bin
Sun Solaris	IBM HTTP Server (IHS)	/usr/lpp/WebSphere/DownloadPlugins/Solaris/IHS/mod_ibm_app_server_http.so_bin
	Lotus Domino	/usr/lpp/WebSphere/DownloadPlugins/Solaris/Domino/libdomino5 _ http.a_bin
	Apache	/usr/lpp/WebSphere/DownloadPlugins/Solaris/Apache/mod_app_server_http.so_bin
	iPlanet (Netscape)	/usr/lpp/WebSphere/DownloadPlugins/Solaris/iPlanet/libns41_ http.so_bin
LINUX	IBM HTTP Server (IHS)	/usr/lpp/WebSphere/DownloadPlugins/LINUX/IHS/mod_ibm_app_server_http.so_bin
	Apache	/usr/lpp/WebSphere/DownloadPlugins/LINUX/Apache/mod_app_server_http.so_bin

For the Windows platform, you must also download the plugin_common.dll file from the /usr/lpp/WebSphere/DownloadPlugins/WIN32 directory.

Note: When downloading these plug-ins, remove the ″_bin″ suffix from the file name. It is included as a reminder that the plug-ins are in binary format. For example, when downloading the mod_ibm_ app_server_http.dll_bin plug-in for use with an IBM HTTP Server on Windows 2000/NT, use mod_ibm_ app_server_http.dll as the file name.

4. Configuring the Web server.

Once the appropriate plug-in files are downloaded to the Web server, update the Web server’s configuration file with the location of the plug-in and plug-in’s configuration file. In addition, for Windows 2000/NT, the location of the common file, plugin_common.dll, must be added to the PATH statement.

Instructions for configuring plug-ins within the Web server configuration file are contained in the Web server’s documentation. An example can also be found in section 6.6.45.0.1 of the WebSphere Application Server Advanced Edition for Distributed Platforms’ InfoCenter, ″Modifications to Web server configuration files during product installation″.

Since the plug-in files are manually downloaded, any changes to the Web server configuration file must be done manually. The changes will look like those listed in the WebSphere Application Server Advanced Edition InfoCenter. For example, for the IBM HTTP Server on Windows 2000/NT, the following changes must be made to the httpd.conf file:

```
LoadModule ibm_app_server_module <download directory>\mod_app_server_http.dll
WebSpherePluginConfig  C:\WebSphere\AppServer\config\plugin-cfg.xml
```

The LoadModule statement tells the Web server where to find the plug-in code. The WebSpherePluginConfig statement is used by the plug-in upon start-up to find its own configuration file. For exact syntax and placement within the Web server configuration file, see your Web server documentation.

5. Use the WebSphere for z/OS Administration application to add `BBOC_HTTP_PORT` and/or `BBOC_HTTP_SSL_PORT` environment variables to the current.env file for each WebSphere for z/OS J2EE server instance to which requests will be redirected. These environment variables must specify the WebSphere for z/OS ports to which the Web server plug-in should redirect these requests. These same ports must be specified on a <Transport Hostname> element in the plug-in's plugin-cfg.xml file.
6. (Optional.) Download GSkit if you want to use SSL with this configuration.

   In addition to the plug-in files, there is another software package provided with WebSphere for z/OS that helps connect distributed platform Web servers to WebSphere for z/OS. This package is required if the Secure Socket Layer (SSL) Transport (also known as HTTPS) is used. This package is called the Global Security Kit, or GSkit. There is one GSkit install image per platform. GSkit is the same for all Web servers running on a platform. The following table identifies where the GSkit install image resides for each platform:

Table 23. Location of Gskit install image

Operating system	Install image file
Windows 2000/NT	/usr/lpp/WebSphere/ DownloadPlugins/ Win32gsk5bas.exe_bin
AIX	/usr/lpp/WebSphere/ DownloadPlugins/AIX/ gsk/gskkm.rte_bin
HPUX	/usr/lpp/WebSphere/ DownloadPlugins/HPUX/ gsk/gsk5bas.tar.Z_bin
Sun Solaris	/usr/lpp/WebSphere/ DownloadPlugins/Solaris/ gsk/gsk5bas.tar.Z_bin
LINUX	/usr/lpp/WebSphere/ DownloadPlugins/LINUX/ gsk/gsk5bas-5.0-4.79. i386.rpm_bin

Notes:

a. The appropriate GSkit install image file should be downloaded from WebSphere for z/OS to the Web server and installed using the native install process on that platform. For example, DSMIT should be run on AIX, or the gsk5bas.exe, which invokes InstallShield, should be run on Windows.
b. The GSkit install directory on the Web server must be added to the PATH statement.
c. When downloading these files, remove the "_bin" suffix from the file name. It is included as a reminder that these files are in binary format. For example, when downloading the gskkm.rte_bin file for the AIX Gskit install image residing on AIX, use gskkm.rte as the file name.
d. If you intend to use SSL, you must also make sure that:
   - Your Web server is configured for this support. (See your Web server documentation for a description of how to configure SSL for your specific Web server if it has not already been configured for this support.)

If you are using an IBM HTTP Server, you must add the following lines to the bottom of your Web server's httpd.conf file:

```
LoadModule ibm_ssl_module modules/IBMModuleSSL128.dll
Listen port_number
Keyfile C:\ssl\http_session\plug-inKeys.kdb
<VirtualHost virtual_host_name:port_number>
      ServerName virtual_host_name
      SSLEnable
      SSLClientAuth none
</VirtualHost<
```

These lines causes the Web server to listen on the specified port.

The "SSLClientAuth none" element indicates that you do not want to enable client authentication. If you want to use client authentication, change this line to:

```
SSLClientAuth enable
```

This will cause the HTTP Server to send a request for a certificate to the browser. Your browser may prompt you to choose a certificate to send to the Web server in order to perform client authentication.

- You have configured an HTTPS Transport Handler and the `BBOC_HTTP_SSL_PORT` environment variable for that Transport Handler specifies the port the Web server plug-in will be using to redirect requests to the WebSphere for z/OS Web container. This same port must be specified on a <Transport Hostname> element in the plug-in's plugin-cfg.xml file. (You can use the Administration application to verify that an HTTPS Transport Handler has been properly configured.)
- You have configured your Web server plug-in for SSL by:
  1) Creating an SSL key file for the Web server plug-in.

     The contents of this file depend on whom you want to allow to communicate directly with WebSphere for z/OS over the port number specified for the HTTPS Transport Handler (in other words, you are defining the HTTPS server security policy). The following procedure describes how to create an SSL key file with a restrictive security policy, in which only a well-defined set of clients (the WebSphere plug-ins for the Web server) are allowed to connect to the WebSphere for z/OS HTTPS Transport Handler:

     a) Create an SSL key file without the default signer certificates.

        i. Start IKeyMan.

           On Windows, start IKeyMan from the WebSphere Application Server entry on the Windows Start menu.

        ii. Create a new key database file.

           Click Key Database File and select New. Then specify settings:

           – Key database type: JKS
           – File Name: appServerKeys.jks
           – Location: your myKeys directory, such as `install_root/myKeys`

           Click OK.

        iii. Enter a password (twice for confirmation) and click OK.

        iv. Delete all of the signer certificates.

        v. Click Signer Certificates and select Personal Certificates.

vi. Add a new self-signed certificate.

Click New Self-Signed to add a self-signed certificate. Then specify settings:

- Key Label: appServerTest
- Organization: IBM

Click OK.

vii. Extract the certificate from this self-signed certificate so that it can be imported into the plug-in's SSL key file.

Click Extract Certificate. Then specify settings:

- Data Type: Base64-encoded ASCII data
- Certificate file name: appServer.arm
- Location: the path to your myKeys directory

Click OK.

viii. Import the plug-in's certificate.

Click Personal Certificates. Select Signer Certificates. Click Add. Then specify settings:

- Data Type: Base64-encoded ASCII data
- Certificate file name: appServer.arm
- Location: the path to your myKeys directory

Click OK.

ix. Enter ″plug-in″ for the label and click OK.

x. Click Key Database File, and select Exit.

b) Add the WebSphere for z/OS signer certificate to the plug-in's SSL key file.

i. Start the key management utility.

ii. Click the Key Database File menu and select Open.

iii. Select the file `install_root`/myKeys/plug-inKeys.kdb.

The plug-inKeys.kdb file is the key database file, created using the z/OS System SSL gskkyman utility, that contains the public keys, private keys, trusted CAs, and certificates for the Web server plug-ins.

iv. Enter the associated password and click OK.

v. Click Personal Certificates and select Signer Certificates.

vi. Click Add. Then specify settings:

- Data Type: Base64-encoded ASCII data
- Certificate File Name: appServer.arm
- Location: the path to your myKeys directory

vii. Click OK.

viii. Click Key Database File and select Exit.

c) Reference the key file in the WebSphere for z/OS Administration application.

Reference the appropriate SSL key file in the default SSL settings configuration panel or in the HTTPS SSL settings configuration panel. Using the default SSL settings panel, you would:

i. Start the administrative console.

ii. Open the Security Center.

iii. Specify settings in the default SSL configuration:

- Key File Name: `install_root/myKeys/appServer.jks`
- Key File Password: enter your password
- Key File Format: JKS
- Trust File Name: (empty)
- Trust File Password: (empty)
- Client Authentication: selected

iv. Save your changes.

2) Modifying the Web server plug-in's configuration file to indicate that you are using an HTTPS Transport Handler and to add the keyring and stashfile properties to the definition of this Transport Handler.

Example: The Server Group definition for J2EE server BBS1122, with server instances BBS1122A and BBS1122B defined, would look like the following:

```
<ServerGroup Name="Default Host/Default Server">
     <Server CloneID="BBS1122.BBS1122A" Name="INSTANCE_A">
         <Transport Hostname="websphere1.ibm.com" Port="8083"
                          Protocol="https"/>
                 <Property name="keyring" value="C:\ssl\http_session
                              \plug-inKeys.kdb">
                 <Property name="stashfile" value="C:\ssl\
                              http_session\plug-inpw.sth">
            </Transport>
     </Server>
     <Server CloneID="BBS1122.BBS1122B" Name="INSTANCE_B">
         <Transport Hostname="websphere1.ibm.com" Port="8084"
                          Protocol="https"/>
                  <Property name="keyring" value="C:\ssl
                               \http_session\plug-inKeys.kdb">
                  <Property name="stashfile" value="C:\ssl
                              \http_session \plug-inpw.sth">
            </Transport>
     </Server>
</ServerGroup>
```

where:

- plug-inKeys.kdb is the key database file, created using the z/OS System SSL gskkyman utility, that contains the public keys, private keys, trusted CAs, and certificates for the Web server plug-ins.file containing the keys for the plug-ins, and
- plug-inpw.sth is the stash file in which the z/OS System SSL gskkyman utility stored the encrypted database password for these certificates.

See your Web server documentation for more information about these files.

7. Configure the plug-in.

The WebSphere plug-ins for Web servers are configured using a plugin-cfg.xml file. Following is an example of a plugin-cfg.xml file. This sample plugin-cfg.xml file contains the following configuration specifications:

- LogLevel, which can be set to ″Warn″, ″Error″, or ″Trace″. ″Trace″ should only be used for debugging purposes. The Name attribute on the LogLevel element specifies the location of the log file.
- VirtualHostGroup provides a logical grouping of a set of host definitions.

- Server CloneID is the key for maintaining session affinity. By default, the Web container assigns a CloneID of <ServerName.ServerInstanceName>.

  If you prefer a different CloneID, you can use the WebSphere for z/OS Administration application to set the CLONEID environment variable in the current.env file. The value specified on this environment variable overrides the default value.

  If you specify a value on the CLONEID environment variable, you must also change the value specified for a Server element in the Web server plug-in's plugin-cfg.xml file to match this value.
- UriGroup provides a logical grouping of a set of URIs.
- Route links the ServerGroup, UriGroup, and VirtualHostGroup definitions together.

On a Windows 2000 system, the default location for this file is `C:\WebSphere\AppServer\config\plugin-cfg.xml`

```
<?xml version="1.0" encoding="UTF-8" ?>
<Config>
          <Log LogLevel="Warn" Name="C:\WebSphere\AppServer\logs\plugin.log"/>
         <VirtualHostGroup Name="default_host">
              <VirtualHost Name="*:80"/>
              <VirtualHost Name="*:9080"/>
         </VirtualHostGroup>
         <ServerGroup Name="Default Host/Default Server">
              <Server CloneID="BBS1122.BBS1122A" Name="INSTANCE_A">
                  <Transport Hostname="websphere1.ibm.com" Port="8083"
                                   Protocol="http"/>
              </Server>
              <Server CloneID="BBS1122.BBS1122B" Name="INSTANCE_B">
                  <Transport Hostname="websphere1.ibm.com" Port="8084"
                                   Protocol="http"/>
              </Server>
       </ServerGroup>
       <UriGroup Name="test_uri_group">
              <Uri Name="/webapp"/>
              <Uri Name="/IntPlusThreeCR/IntPlusThree"/>
              <Uri Name="/catcher/servlet/HelloSessionServlet"/>
      </UriGroup>
      <Route ServerGroup="Default Host/Default Server"
           UriGroup="test_uri_group" VirtualHostGroup="default_host"/>
</Config>
```

Additional elements and attributes that can be included in a plugin-cfg.xml file are described in "Properties of WebSphere plug-ins for Web servers" on page 183..

---

8. (Required only if you intend to use private headers.) Using the Administration application, add the `BBOC_HTTP_MODE=INTERNAL` and/or `BBOC_HTTP_SSL_MODE=INTERNAL` environment variables to the current.env file for a specific J2EE server instance. These environment variables enable the HTTP Transport Handler to trust private headers received from the Web server's plug-in, over the port specified on the `BBOC_HTTP_PORT` and/or `BBOC_HTTP_SSL_PORT` environment variables.

   Notes:

   a. If you try to use private headers without adding this variables to the current.env file, private headers will be ignored. If the private headers are ignored, WebSphere for z/OS might not be able to locate the requested application.

b. If you add these environment variable to the current.env file, the HTTP Transport Handler will trust all private headers it receives. Therefore, you must ensure that there are no untrusted paths to the HTTP Transport Handler.

---

9. Stop WebSphere for z/OS and the Web server and and start them again.

   The configuration is complete. In order to activate the configuration, stop and restart both the Web server and the application server.

Installing a Web server plug-in on a Microsoft Internet Information Server (IIS)

The following additional steps must be performed when installing a Web server plug-in on a Microsoft Internet Information Server (IIS) Version 4.0 or 5.0:

1. Start the Internet Service Manager application.
2. Create a new Virtual Directory for the Web site instance you want to work with WebSphere Application Server. To do this with a default installation:
   a. Expand the tree on the left until you see ″Default Web Site″. Right click on ″Default Web Site″ and select ″′New″->″Virtual Directory″. This will bring up the wizard for adding a virtual directory.
   b. In the space provided for ″Alias to be used to Access Virtual Directory″, type ′sePlugins′.
   c. In the space provided for ″Enter the physical path of the directory containing the content you want to publish″, browse into the WebSphere bin directory.
   d. For ″What access permissions do you want to set for this directory″, check ″Allow Execute Access″.
   e. Click finish. A virtual directory, titled ″sePlugins″, should be added to you default Web site.
3. Add the ISAPI filter into the IIS configuration.
   a. Right click on the hostname in the tree on the left and select **Properties**.
   b. On the **Internet Information Services** tab, select **WWW Service** in the **Master Properties** drop down box and click on the **Edit** button. The **WWW Service Master Properties** window should pop up.
   c. Click on the **ISAPI Filters** tab.
   d. Click on the **Add** button. This should bring up the **Filter Properties** window.
   e. In the space provided next to **Filter Name:**, type **iisWASPlugin**.
   f. In the space provided next to **Executable**, click the browse button and go into the **usr//lpp/WebSphere/ DownloadPlugins/Win32/ IIS/iisWASPlugin _http.dll_bin** directory and select **iisWASPlugin_http.dll**.
   g. Click the **OK** button until all open windows are closed.
4. Add the variable **Plugin Config** to the registry under the path **HKEY_LOCAL_MACHINE -> SOFTWARE -> IBM -> WebSphere Application Server -> 4.0**. Set the value for this variable to the location of the plugin-cfg.xml file.

Properties of WebSphere plug-ins for Web servers

The plugin-cfg.xmlfile includes the following elements and attributes:

Config (exactly one)

This element starts the WebSphere HTTP plug-in configuration file. It contains all other elements and attributes of the configuration.

This section resembles the following in the file:

```
<Config IgnoreDNSFailures="true" RefreshInterval="240">
```

IgnoreDNSFailures (zero or one attribute for each Config)
: Specifies whether the plug-in ignores DNS failures within a configuration when starting. When set to true, the plug-in ignores DNS failures within a configuration and starts successfully if at least one server in each ServerCluster is able to resolve the host name. The server is marked unavailable for the life of the configuration. No attempts to resolve the host name are made later on during the routing of requests. If a DNS failure occurs, a log message is written to the plug-in log file and the plug-in initialization continues rather than causing the Web server not to start. The default value is false, meaning DNS failures cause the Web server not to start.

Refresh interval (zero or one attribute for each Config)
: The time interval (in seconds) at which the plug-in should check the configuration file to see if updates or changes have occured. The plug-in checks the file for any modifications that have occurred since the last time the plug-in configuration was loaded.

  In a development environment in which changes are frequent, a lower setting than the default setting of 60 is preferable. In production, a higher value then the default is preferable because updates to the configuration will not occur so often. If the plug-in reload fails for some reason, a message is written to the plug-in log file and the previous configuration is used until the plug-in config reloads successfully. If you are not seeing the changes you made to your plug-in configuration, check the plug-in log file for indications of the problem.

ASDisableNagle (zero or one attribute for each Config)
: Specifies whether the user wants to disable nagle algorithm for the connection between the plug-in and the application server. By default, nagle algorithm is enabled.

  The value can be true or false.

IISDisableNagle (zero or one attribute for each Config)
: he nagle algorithm is enabled by default on Microsoft Internet Infomations Services (IIS). This attribute can be used to disable the nagle algorithm.

  The value can be true or false.

ResponseChunkSize (zero or one attribute for each Config)
: The plug-in reads the response body in 64k chunks until all of the response data is read. This causes a performance problem for requests whose response body contains large amounts of data.

  The ResponseChunkSize attribute allows the users to specify the maximum chunk size to use when reading the response body. For example, <Config ResponseChunkSize=″N″>, where N equals the chunk size in kilobytes.

  If the content length of the response body is unknown, a buffer size of N kilobytes is allocated and the body is read in N kilobyte size chunks, until the entire body is read. If the content length is known, then a buffer size of either content length or N (whichever is less) is used to read the response body.

The default chunk size is 64k.

Log (zero or one element for each Config)

The log describes the location and level of log messages that are written by the plug-in. If a log is not specified within the config, then, in some cases, log messages are written to the Web server error log.

This section resembles the following in the file:

```
<Log LogLevel="Error" Name="log_directory/filename"/>
```

Name (exactly one attribute for each Log)

The level of detail of the log messages that the plug-in should write to the log. Values for this attribute are one of the following:

- Trace
- Warn
- Error

If a LogLevel is not specified with the Log, then the default value is Error.

Trace allows you to see the steps in the request process in detail. Warn and Error means that only information about abnormal request processing will be logged.

Be careful when setting the level to Trace. A lot of error messages are logged at this level which can cause the file system to fill up very quickly. A Trace setting should never be used in a normally functioning environment as it affects performance.

ServerCluster (one or more elements for each Config)

A group of servers that are generally configured to service the same types of requests.

In the simplest case, the cluster contains only one server definition. In the case in which more than one server is defined, the plug-in will load balance across the defined servers using either a Round Robin or a Random algorithm. The default is Round Robin.

This section resembles the following in the file:

```
<ServerCluster Name="Servers">
<ClusterAddress Name="ClusterAddr">
<Transport Hostname="192.168.1.2" Port="9080" Protocol="HTTP"/>
<Transport Hostname="192.168.1.2" Port="9443" Protocol="HTTPS">
<Property Name="Keyring" value="c:/WebSphere/AppServer/keys/keyring.kdb"/>
<Property Name="Stashfile" value="c:/WebSphere/AppServer/keys/keyring.sth"/>
</ClusterAddress>
<Server Name="Server1">
<Transport Hostname="192.168.1.3" Port="9080" Protocol="HTTP"/>
<Transport Hostname="192.168.1.3" Port="9443" Protocol="HTTPS">
<Property Name="Keyring" value="c:/WebSphere/AppServer/keys/keyring.kdb"/>
<Property Name="Stashfile" value="c:/WebSphere/AppServer/keys/keyring.sth"/>
</Server>
<Server Name="Server2">
<Transport Hostname="192.168.1.4" Port="9080" Protocol="HTTP"/>
<Transport Hostname="192.168.1.4" Port="9443" Protocol="HTTPS">
<Property Name="Keyring" value="c:/WebSphere/AppServer/keys/keyring.kdb"/>
<Property Name="Stashfile" value="c:/WebSphere/AppServer/keys/keyring.sth"/>
</Server>
<Server Name="Server3">
<Transport Hostname="192.168.1.5" Port="9080" Protocol="HTTP"/>
<Transport Hostname="192.168.1.5" Port="9443" Protocol="HTTPS">
<Property Name="Keyring" value="c:/WebSphere/AppServer/keys/keyring.kdb"/>
```

```
<Property Name="Stashfile" value="c:/WebSphere/AppServer/keys/keyring.sth"/>
</Server>
<PrimaryServers>
<Server Name="Server1"/>
<Server Name="Server2"/>
</PrimaryServers>
<BackupServers>
<Server Name="Server3"/>
</BackupServers>
</ServerCluster>
```

Note: If you are using the WebSphere HTTP Plug-in for z/OS, the Property Name=keyring and the Property Name=stashfile elements included here will be ignored if they are included in the plugin-cfg.xml file for that plug-in. The WebSphere HTTP Plug-in for z/OS uses the SSL setup specified in the hosting HTTP Server's httpd.conf file and does not look for these elements in the plugin-cfg.xml file.

Name (exactly one attribute for each ServerCluster)
The logical or administrative name to be used for this group of servers.

LoadBalance (zero or one attribute for each ServerCluster)
The default load balancing type is Round Robin.

The Round Robin implementation has a random starting point. This means that the first server picked will be done so randomly and then round robin will be used from that point forward. This is so that in multiple process based Web servers all of the processes don't start up by sending the first request to the same application server.

RetryInterval (zero or one attribute for each ServerCluster)
An integer specifying the length of time that should elapse from the time that a server is marked down to the time that the plug-in will retry a connection. The default is 60 seconds.

RemoveSpecialHeaders (zero or one attribute for each ServerCluster)
The plug-in adds special headers to the request before it is forwarded to the application server. These headers store information about the request that will need to be used by the application. By default the plug-in will remove these headers from incoming requests before adding the headers it is supposed to add.

The value can be true or false. Setting the attribute to false introduces a potential security exposure by not removing headers from incoming requests.

CloneSeparatorChange (zero or one attribute for each ServerCluster
Some pervasive devices cannot handle the colon character (:) used to separate clone IDs in conjunction with session affinity. This attribute for the server group tells the plug-in to expect the plus character (+) as the clone separator. You must change application server configurations so that an application server separates clone IDs with the plus character as well.

The value can be true or false.

PostSizeLimit (zero or one attribute for each ServerCluster)
The maximum number of bytes of request content allowed in order for the plug-in to attempt to send the request to an application

server. If a request is received that is greater than this size, the plug-in fails the request. The default value is 10 million bytes.

Server (one or more elements for each ServerCluster)
A WebSphere Application Server instance that is configured to handle requests routed to it given the routing rules of the plug-in configuration. The Server should correspond to an application server running on either the local machine or a remote machine.

Name (exactly one attribute for each Server)
The administrative or logical name for the server.

CloneID (zero or one attribute for each Server)
f this unique ID is present in the HTTP Cookie header of a request (or the URL if using URL rewriting), the plug-in routes the request to this particular server, provided all other routing rules are met. If a CloneID is not specified in the Server, then session affinity is not enabled for this server.

This attribute is used in conjunction with session affinity. When this attribute is set, the plug-in checks the incoming cookie header or URL for **JSESSIONID**. If **JSESSIONID** is found then the plug-in looks for one or more clone IDs. If clone IDs are found and a match is made to this attribute then the request is sent to this server rather than load balanced across the cluster.

If you are not using session affinity then it is best to remove these clone IDs from the configuration because there is added request processing in the plug-in when these are set. If clone IDs are not in the plug-in then it is assumed that session affinity is not on and the request is load balanced across the cluster.

WaitForContinue (zero or one attribute for each Server)
Specifies whether to use the HTTP 1.1 100 Continue support before sending the request content to the application server. Possible attribute values are true or false. The default value is false; the plug-in does not wait for the 100 Continue response from the application server before sending the request content because it is a performance hit.

Enable this function (set to true) when configuring the plug-in to work with certain types of proxy firewalls.

LoadBalanceWeight (zero or one attribute for each Server)
The weight associated with this server when the plug-in does weighted round robin load balancing. The algorithm for this attribute decrements all weights within the server cluster until all weights reach zero. Once a particular server's weight reaches zero, no more requests are routed to that server until all servers in the cluster have a weight of zero. After all servers reach zero, the weights for all servers in the cluster are reset and the algorithm starts over.

ConnectTimeout (zero or one attribute for each Server)
The ConnectTimeout attribute of a Server element enables

the plug-in to perform non-blocking connections with the application server. Non-blocking connections are beneficial when the plug-in is unable to contact the destination to determine if the port is available or unavailable.

If no ConnectTimeout value is specified, the plug-in performs a blocking connect in which the plug-in sits until an operating system times out (as long as 2 minutes depending on the platform) and allows the plug-in to mark the server *unavailable*. A value of 0 causes the plug-in to perform a blocking connect. A value greater than 0 specifies the number of seconds you want the plug-in to wait for a sucessful connection. If a connection does not occur after that time interval, the plug-in marks the server *unavailable* and fails over to one of the other servers defined in the cluster.

ExtendedHandshake (zero or one attribute for each Server)

The ExtendedHandshake attribute is used when a proxy firewall is between the plug-in and the application server. In such a case, the plug-in is not failing over, as expected.

The plug-in marks a server as down when the connect() fails. However, when a proxy firewall is in between the plug-in and the application server, the connect() will succeed, even though the back end application server is down. This causes the plug-in to not failover correctly to other application servers.

The plug-in performs some handshaking with the application server to ensure that it is started before sending the request. This enables the plug-in to failover in the event the application server is down.

The value can be true or false.

Transport (one or more elements for each Server)

The transport for reading and writing requests to a particular WebSphere Application Server instance. The transport provides the information needed to determine the location of the application server to which the request will be sent. If the Server has multiple transports defined to use the same protocol, the first one will be used.

It is possible to configure the Server to have one non-secure transport and one that uses SSL. In this configuration, a match of the incoming request protocol will be performed to determine the appropriate transport to use to send the request to the application server.

Hostname (exactly one attribute for each Transport)

The host name or IP address of the machine on which the WebSphere application server instance is running.

By default, the MaxConnections is set to zero. If the value is zero, then, there is no limit to the number of pending connections to the application servers.

Port (exactly one attribute for each Transport)

The port on which the WebSphere application server instance is listening.

Protocol (exactly one attribute for each Transport)
The protocol to use when communicating over this transport -- either HTTP or HTTPS.

Property (zero, one, or more elements for each Transport)
When the Protocol of the Transport is set to HTTPS, use this element to supply the various initialization parameters, such as password, keyring and stashfile.

Name (exactly one attribute for each Property)
The name of the Property being defined. Supported names recognized by the transport are keyring, stashfile, and password.

Note: password is the only name that can be specified for the WebSphere HTTP Plug-in for z/OS. keyring, and stashfile, if specified, will be ignored.

Value (exactly one attribute for each Property)
The value of the Property being defined.

ClusterAddress (zero or one element for each ServerCluster)
A ClusterAddress is like a Server element in that you can specify the same attributes and elements as for a Server element. The difference is that you can only define one of them within a ServerCluster. Use a ClusterAddress when you do not want the plug-in to perform any type of load balancing because you already have some type of load balancer in between the plug-in and the application server.

If a request comes in that does not have affinity established, the plug-in routes it to the ClusterAddress, if defined. If affinity has been established, then the plug-in routes the request directly to the clone, bypassing the ClusterAddress entirely. If no ClusterAddress is defined for the ServerCluster, then the plug-in load balances across the PrimaryServers list.

PrimaryServers (zero or one element for each ServerCluster)
Lists defined servers to which the plug-in routes requests for this cluster. If a list of PrimaryServers is not specified, the plug-in routes requests to servers defined for the ServerCluster.

BackupServers (zero or one element for each ServerCluster)
Lists servers to which requests should be sent to if all servers specified in the PrimaryServers list are unavailable. The plug-in does not load balance across the BackupServers list but traverses the list in order until no servers are left in the list or until a request is successfully sent and a response received from an application server.

VirtualHostGroup (zero, one, or more elements for each Config)
A group of virtual host names that will be specified in the HTTP Host header. Enables you to group virtual host definitions together that are configured to handle similar types of requests.

This section resembles the following in the file:

```
<VirtualHostGroup Name="Hosts">
<VirtualHost Name="www.x.com"/>
<VirtualHost Name="www.x.com:443"/>
```

```
<VirtualHost Name="*:8080"/>
<VirtualHost Name="www.x.com:*"/>
<VirtualHost Name="*:*"/>
</VirtualHostGroup>
```

Name (exactly one attribute for each VirtualHostGroup)
The logical or administrative name to be used for this group of virtual hosts.

VirtualHost (one or more elements for each VirtualHost)
The name used for a virtual or real machine used to determine if incoming requests should be handled by WebSphere Application Server or not. Use this element to specify host names that will be in the HTTP Host header which should be seen for requests that need to be handled by the application server. You can specify specific host names and ports that incoming requests will have or specify an * for either the host name, port, or both.

Name (exactly one attribute for each VirtualHost)
The actual name that should be specified in the HTTP Host header in order to match successfully with this VirtualHost.

The value is a host name or IP address and port combination, separated by a colon.

You can configure the plug-in to route requests to the application server based on the incoming HTTP Host header and port for the request. The Name attribute specifies what those combinations are.

You can use a wildcard for this attribute. The only acceptable solutions are either an * for the host name, an * for the port, or an * for both. An * for both means that any request will match this rule. If no port is specified in the definition the default HTTP port of 80 is assumed.

UriGroup (zero, one or more elements for each Config)
A group of URIs that will be specified on the HTTP request line. The same application server must be able to handle the URIs. The route will compare the incoming URI with the URIs in the group to determine if the application server will handle the request.

This section resembles the following in the file:

```
<UriGroup Name="Uris">
<Uri Name="/servlet/snoop"/>
<Uri Name="/webapp/*"/>
<Uri Name="*.jsp"/>
</UriGroup>
```

Name (exactly one attribute for each UriGroup)
The logical or administrative name for this group of URIs.

Uri (one or more elements for each UriGroup)
The virtual path to the resource that will be serviced by WebSphere Application Server. Each URI specifies the incoming URLs that need to be handled by the application server. You can use a wildcard in specify these definitions.

Name (exactly one attribute for each Uri)
The actual string that should be specified in the HTTP request line in order to match successfully with this URI. You can use a wildcard within the URI definition. You can specify rules such as *.jsp or /servlet/* to be handled by WebSphere Application Server. When you assemble your application, if you specify **File Serving**

Enabled then only a wildcard URI is generated for the Web application, regardless of any explicit servlet mappings. If you specify **Serve servlets by classname** then a URI having <Uri Name="*Web_application_URI*/servlet/*"> is generated.

AffinityCookie (zero or one attribute for each Uri)
The name of the cookie the plug-in should use when trying to determine if the inbound request has session affinity. (See the description of the CloneID attribute for additional session affinity information.)

Route (one or more elements for each Config)
A request routing rule by which the plug-in will determine if an incoming request should be handled by a WebSphere application server.

The route definition is the central element of the plug-in configuration. It specifies how the plug-in will handle requests based on certain characteristics of the request. The route definition contains the other main elements: a required ServerCluster, and either a VirtualHostGroup, UriGroup, or both.

Using the information that is defined in the VirtualHostGroup and the UriGroup for the route, the plug-in determines if the incoming request to the Web server should be sent on to the ServerCluster defined in this route.

This section resembles the following in the file:

```
<Route VirtualHostGroup="Hosts" UriGroup="Uris" ServerCluster="servers/>
```

VirtualHostGroup (zero or one attribute for each Route)
The group of virtual hosts that should be used in route determination. The incoming host header and server port are matched to determine if this request should be handled by the application server.

It is possible to omit this from the route definition. If it is not present then every request will match during the virtual host match portion of route determination.

UriGroup (zero or one attribute for each Route)
The group of URIs to use for determining the route. The incoming URI for the request is matched to the defined URIs in this group to determine if this request should be handled by the application server.

It is possible to omit this from the route definition. If it is not present than every request will match during the URI match portion of route determination.

ServerCluster (exactly one attribute for each Route)
The cluster to which to send request that successfully match the route.

The cluster that should be used to handle this request. If both the URI and the virtual host matching is successful for this route then the request is sent to one of the servers defined within this cluster.

RequestMetrics (zero or one element for each Config)
This element is used to determine if request metrics is enabled, and how to filter the requests based on the Intenet protocol (IP) and Uniform Resource Identifiers (URI) filters when request metrics is enabled.

This section resembles the following in the file:

```
<RequestMetrics armEnabled="false" newBehavior="false"
   rmEnabled="false" traceLevel="PERF_DEBUG">
```

armEnabled (zero or one attribute for RequestMetrics)
: This attribute is currently ignored by the plug-in. It is for the future usage.

newBehavior (zero or one attribute for RequestMetrics)
: This attribute is currently ignored by the plug-in. It is for the future usage.

rmEnabled (exactly one attribute for RequestMetrics)
: This attribute indicates whether or not the request metrics is enabled in the plug-in. When it is true, the plug-in request metrics will look at the filters and log the request trace record in the plug-in log file. This is performed if a request passes the filters.

traceLevel (exactly one attribute for RequestMetrics)
: When rmEnabled is true, this attribute indicates how much information is logged. When it is NONE, there is no request logging. When it is not NONE, the request response time (and other request information) is logged when the request is done.

filters (zero, one, or two attributes for RequestMetrics)
: When rmEnabled is true, the filters control which requests are traced.

enable (exactly one attribute for each filter)
: When enable is true, the type of filter is on and requests must pass the filter.

type (exactly one attribute for each filter)
: There are two types of filters: SOURCE_IP (for example., client IP address) and URI. For the SOURCE_IP filter type, requests are filtered based on a known IP address. You can specify a mask for an IP address using the asterisk (*). If the asterisk is used, the asterisk must always be the last character of the mask, for example 127.0.0.*, 127.0.*, 127*. For performance reasons, the pattern matches character by character, until either an asterisk is found in the filter, a mismatch occurs, or the filters are found as an exact match.

  For the URI filter type, requests are filtered based on the URI of the incoming HTTP request. The rules for pattern matching are the same as matching SOURCE_IP address filters.

  If both URI and client IP address filters are enabled, Request Metrics requires a match for both filter types. If neither is enabled, all requests are considered a match.

filterValues (one or multiple attribute for each filter)
: The filterValues show the detailed filter information.

value (exactly one attribute for each filterValue)
: Specifies the filter value for the corresponding filter type. This could be either a client IP address or a URI.

Steps for enabling a custom user registry

A custom user registry is enabled on a J2EE server environment basis. For each J2EE server environment in which you want to use a custom user registry, perform the following steps :

1. Make sure the WebSphere CustomRegistry interface was used to encapsulate the custom user registries you will be using. Because the WebSphere CustomRegistry interface is identical across all IBM WebSphere Application Server platforms, if you used the same registries on another IBM WebSphere Application Server platform, they should already be properly encapsulated, and you can go to Step 2. Otherwise, see "Implementing the CustomRegistry interface" on page 199 for a description of how to encapsulate your registries.

2. Make the following changes to the webcontainer.conf configuration file:
   - Add the `WebAuth.CustomRegistry.ImplClass=` `com.`*`company_name`*`.`*`implementation_class_name`* property to enable the CustomRegistry interface.
   - Add the `WebAuth.CustomRegistry.Properties=`*filename*`.properties` property to specify the name of the custom registry properties file. *filename* is the fully qualified name of this properties file.
   - Add the `WebAuth.CustomRegistry.authorizationTableXML=`*filename*`.xml` property to specify the name of the XML file containing a list of the XML files containing authorization tables that the J2EE server will use to authenticate and authorize users and groups. *filename* is the fully qualified name of this file. See "Creating the XML file that defines the location of a Web application's authorization table" on page 194 for an example of this file.
   - Add the `WebAuth.CustomRegistry.SAFPrincipal=`*validMVSUserid* property to specify the MVS user ID under which requests from custom registry clients are processed. (For example, this ID will be used when processing EJB requests or to access an MVS resource that has a deployment descriptor containing the attribute res-auth=container.) This property is only required if you do not want to use the J2EE server's ID to establish these connections.
   - Update the `WebAuth.UnauthenticatedUserSurrogate=` property with the CustomUserRegistry user ID under which unauthenticated clients are to be executed.
   - Make sure the following properties are set if your custom user registry is going to be using single sign-on support:
     - Make sure the `WebAuth.SingleSignOn.Enabled` property is set to true.
     - Make sure the `WebAuth.SingleSignOn.Domain` property is set to NULL.
   - Make sure the following properties are set if your custom user registry is going to be using Form-Based logon support:
     - Make sure the `WebAuth.LoginToken.Encrypt` property is set to true.
     - Make sure the `WebAuth.LoginToken.EncryptionKeyLabel` property is set to the appropriate ICSF key label.
     - Make sure the `WebAuth.LoginToken.LimitToSecureConnections` property is set to true.

   These properties are described in more detail in Appendix B, "Default webcontainer.conf file," on page 351.

3. Use the Administration application to make the following changes to the J2EE server's environment variables:
   - Add the custom user registry implementation class to the `WS_EXT_DIRS` environment variable.
   - (Optional.) Add the `ENABLE_TRUSTED_APPLICATIONS` environment variable to your current.env file and set it to 1. This property is only required if the `WebAuth.CustomRegistry.SAFPrincipal=` webcontainer.conf file property is set to an MVS ID other than the ID of the J2EE server.

     When this variable is set to 1, RACF checks if the client is authorized, and if the client is authorized, the call is successful. If the `WebAuth.CustomRegistry.SAFPrincipal=` property is set to an MVS ID other than the ID of the J2EE server, and the `ENABLE_TRUSTED_APPLICATIONS` environment variable is not set to 1, the client will get an exception.

---

4. Make the following changes to the JVM properties files:
   - Set the `WEB_SECURITY_VERSION` property to 2 to enable Web Security Collaborator Version 2. When Version 2 is enabled, incoming requests can be selectively authorized and refused.
   - Add the `com.ibm.websphere.security.AuthorizationTable` property and use it to specify the class for the authorization table implementation. The default implementation class provided with the product is:

     ```
     com.ibm.ws390.wc.security.AuthorizationTableImpl
     ```

---

5. Add the XML file specified on the `WebAuth.CustomRegistry.authorizationTableXML=`*filename*`.xml` property to the system on which WebSphere for z/OS is running. (See "Creating the XML file that defines the location of a Web application's authorization table" for an example of this file.) This files must be in ASCII format.
6. Add the XML files containing the authorization tables to the system on which WebSphere for z/OS is running. These XML files define the roles that are authorized to use specific applications. (See "Creating XML files containing authorization tables" on page 196 for an example of this file.) This files must be in ASCII format.
7. Stop the J2EE server and start it again.

Creating the XML file that defines the location of a Web application's authorization table

The XML file defining the location of the authorization tables for Web applications can be placed in any directory on your WebSphere for z/OS system. However, you must include the fully qualified name of this file on the `WebAuth.CustomRegistry.authorizationTableXML` property in the webcontainer.conf file. When creating this file, you must:

- Encode it in ASCII format.
- Start it with a <AuthTableList> tag and end it with a </AuthTableList> tag.
- Include the following tag set for every application installed on the J2EE server for which the custom user registry will be used to authenticate and authorize clients:

```
<Application>
      <name>application_name</name>
      <permission-file-name>fully_qualified_file_name
      </permission-file-name>
</Application>
```

Notes:

1. The value specified on the <name> tag is case sensitive. Therefore any name specified on this tag must **exactly** match the name of a Web application installed on the J2EE server as it appear on the Administration application panel.
2. The value specified on the <permission-file-name> tag is also case sensitive. If the XML file specified on a <permission-file-name> tag for an application is not found or is misspelled, the application request will not be processed unless you set up a default authorization table to handle these situations. To set up a default authorization table, include the following tags to point to the XML file containing that table:

   ```
   <Application>
         <name>*</name>
         <permission-file-name>file_name</permission-file-name>
   </Application>
   ```
   and then include "*" on an <application appName> tag in that XML file.

Following is an example of an XML file that defines where the authorization table for each of the four applications installed on a J2EE sever resides. In this example:

- The authorization information for the applications SecureFormWebapp, SecureBasicWebapp1, and SecureBasicWebapp2 is contained in the file authTable3.xml.
- The authorization information for the applications PayrollWebapp and EmployeeDirectory is contained in the file PayrollTable.xml.
- The default authorization table is contained in the file authTable3.xml.

```
<AuthTableList>
   <Application>
      <name>SecureFormWebapp</name>
      <permission-file-name>/webapps/apps0806/authTable3.xml
      </permission-file-name>
   </Application>
   <Application>
      <name>SecureBasicWebapp1</name>
      <permission-file-name>/webapps/apps0806/authTable3.xml
      </permission-file-name>
   </Application>
   <Application>
      <name>SecureBasicWebapp2</name>
      <permission-file-name>/webapps/apps0806/authTable3.xml
      </permission-file-name>
   </Application>
   <Application>
      <name>PayrollWebapp</name>
      <permission-file-name>/webapps/apps0806/payrollInfo.xml
      </permission-file-name>
   </Application>
   <Application>
      <name>EmployeeDirectory</name>
      <permission-file-name>/webapps/apps0806/payrollInfo.xml
      </permission-file-name>
   </Application>
   <Application>
```

```
        <name>*</name>
        <permission-file-name>/webapps/apps0806/authTable3.xml
        </permission-file-name>
</Application></AuthTableList>
```

Whenever the J2EE server using this file cannot find an XML file containing the authorization table for a particular application, it will use the role mappings defined in the authorization table for application ″*″ that is contained in the file authTable3.xml. (The section “Creating XML files containing authorization tables” includes an example of role mappings that might exist in files authTable3.xml and PayrollInfo.xml. The example of file authTable3.xml includes a default authorization table.)

Creating XML files containing authorization tables

XML files containing the authorization tables the J2EE server will use to authenticate and authorize clients can be placed in any directory on your WebSphere for z/OS system. However, you must be sure to include the fully qualified name of the files within the appropriate <application> tags in the XML file specified on the `WebAuth.CustomRegistry.authorizationTableXML` property in the webcontainer.conf file. When creating these files you must:

- Encode them in ASCII format.
- Start them with an <authorizationTable> tag and end them with an </authorizationTable> tag.
- Include the following tag set for each application for which authorization information is being included in one of these files:

```
<application>
   <authorizations>
      <role roleName="role_name">
         <group groupName= "group_name"/>
         <user userName="user_name"/>
      </role>
   </authorizations>
</application>
```

<application appName>
This tag is used to specify the name of the enterprise applications for which the authorizations that follow apply.

Notes:

1. If the same users and groups are authorized to use multiple enterprise applications, you can list all of these applications on a single tag. As illustrated in the following example, the names of the Web applications must be separated with a comma and a blank space:

   ```
   <application appName="SecureBasicWebapp1, SecureBasicWebapp2">
   ```

2. The names specified on this tag must exactly match the names of Web applications installed on the J2EE server as they appear on the Administration application panel. The values specified on this tag are case sensitive.

<role roleName>
This tag is used to define the J2EE security role to which the user and group mappings that follow apply. The roles specified on this tag must match the roles defined in the application’s deployment descriptors.

Note: The value specified on this tag is case sensitive. Include multiple versions of this tag if you want to define multiple security roles for the same application.

<group groupName>
This tag is used to specify a user group, any member of which will be mapped to the role specified on the corresponding <role roleName> tag. The group name specified on this tag must exactly match a name defined in the custom user registry.

Note: The value specified on this tag is case sensitive. Include multiple versions of this tag if you want to give multiple groups access to the same application.

<user userName>
This tag is used to specify a specific user ID that will be mapped to the role specified on the corresponding <role roleName> tag. The user name must exactly match a name defined in the custom user registry. The value specified on this tag is case sensitive.

Notes:

1. Include multiple versions of this tag if you want to give multiple users access to the same application.
2. For applications that can be accessed by any successfully authenticated client, you can specify the following <role roleName> tag set in the authorization table for those applications:
```
<role roleName="AuthenticatedUsers">
        <user userName="AllAuthenticatedUsers"/>
</role>
```
3. For applications that can be accessed by any client, you can specify the following <role roleName> tag set in the authorization table for those applications:
```
<role roleName="AllUsers">
        <user userName="Everyone"/>
</role>
```

Multiple <role> tag sets can be included for each application.

Note: If you need to FTP any of these files between J2EE servers, the FTP must be performed in binary format.

Following is an example of the two XML files, authTable3.xml file and payrollInfo.xml that were defined in the example contained in the section, "Creating the XML file that defines the location of a Web application's authorization table" on page 194.

Example of the authTable3.xml file: In this example, authTable3.xml file includes a default authorization table (<application appName="*">) that the J2EE server will use to authenticate and authorize a client if it can not locate the authorization table for a requested application. Also, since the roles defined for application SecureBasicWebapp1 are the same as the roles defined for application SecureBasicWebapp2, both applications are listed on the same <application> tag.

```
<authorizationTable>
<application appName="SecureFormWebapp">
   <authorizations>
      <role roleName="manager">
         <group groupName= "manager"/>
         <group groupName= "super_manager"/>
         <user userName="SP"/>
      </role>
      <role roleName="employee">
         <group groupName= "employee"/>
         <user userName="Everyone"/>
      </role>
      <role roleName="bigboss">
         <user userName="SP"/>
      </role>
   </authorizations>
</application>

<application appName="SecureBasicWebapp1, SecureBasicWebapp2">
   <authorizations>
      <role roleName="manager">
         <group groupName="App1manager"/>
         <group groupName="App1super_manager"/>
         <user userName="App1SP"/>
      </role>
      <role roleName="employee">
         <group groupName="App1employee"/>
         <user userName="Everyone"/>
      </role>
      <role roleName="bigboss">
         <user userName="App1SP"/>
      </role>
   </authorizations>
</application>
<application appName="*">
   <authorizations>
      <role roleName="manager">
         <group groupName="App1super_manager"/>
         <user userName="App1SP"/>
      </role>
      <role roleName="bigboss">
         <user userName="App1SP"/>
      </role>
   </authorizations>
</application>
</authorizationTable>
```

example of the payrollInfo.xml file: In this example, the authorization information for application EmployeeDirectory indicates that a client does not have to be authenticated in order to access this application. Similarly, the authorization information for application PayrollWebapp indicates that any authorized client can access this application.

```
<authorizationTable>
<application appName="EmployeeDirectory">
   <authorizations>
      <role roleName="Public">
         <user userName="Everyone"/>
      </role>
   </authorizations>
</application>

<application appName="PayrollWebapp">
   <authorizations>
      <role roleName="Employee">
         <user userName="AllAuthenticatedUsers"/>
      </role>
   </authorizations>
</application>
</authorizationTable>
```

Implementing the CustomRegistry interface

Use the Administration application to add the implementation class for the CustomRegistry interface to the classpath environment variable in the current.env file. Remember the entire classpath must be entered on the same line.

After you add the name of this file to the current.env classpath, when you activate a conversation or prepare for a cold start, WebSphere for z/OS will write the environment data to an HFS file for each server instance.

The CustomRegistry interface is located in the Java package com.ibm.websphere.security. A description of this interface is available at URL:

`http://www.ibm.com/software/webservers/appserv/doc/v40/ae/apidocs/index.html`

Most of the methods in this interface return either strings or lists. When you implement these methods, indicate failure to retrieve the desired information by returning null strings or null lists.

Following is the structure of the CustomRegistry interface:

```
package com.ibm.websphere.security;

import java.util.*;

public interface CustomRegistry
{

// General methods
public void initialize(java.util.Properties props)
throws CustomRegistryException;

public String getRealm()
throws CustomRegistryException;

// User-related methods
public boolean isValidUser(String userName)
throws CustomRegistryException;

public List getUsers()
throws CustomRegistryException;

public List getUsers(String pattern)
throws CustomRegistryException;

public String getUniqueUserId(String userName)
throws CustomRegistryException,
EntryNotFoundException;

public String getUserSecurityName(String uniqueUserId)
throws CustomRegistryException,
EntryNotFoundException;

public String getUserDisplayName(String securityName)
throws CustomRegistryException,
EntryNotFoundException;

public List getUsersForGroup(String groupName)
throws CustomRegistryException,
EntryNotFoundException;

public List getUniqueUserIds(String uniqueGroupId)
throws CustomRegistryException,
EntryNotFoundException;
// Group-related methods
public boolean isValidGroup(String groupName)
throws CustomRegistryException;

public List getGroups()
throws CustomRegistryException;

public List getGroups(String pattern)
throws CustomRegistryException;

public String getUniqueGroupId(String groupName)
throws CustomRegistryException,
EntryNotFoundException;

public String getGroupSecurityName(String uniqueGroupId)
throws CustomRegistryException,
EntryNotFoundException;

public String getGroupDisplayName(String groupName)
throws CustomRegistryException,
EntryNotFoundException;

public List getGroupsForUser(String userName)
throws CustomRegistryException,
EntryNotFoundException;

public List getUniqueGroupIds(String uniqueUserId)
throws CustomRegistryException,
EntryNotFoundException;
// Authentication methods
public String checkPassword(String userId, String password)
throws PasswordCheckFailedException,
CustomRegistryException;

}
```

Steps for pre-compiling JSPs

You can use the IBM provided JspBatchCompiler.sh script to pre-compile JSPs before they are requested, thus improving performance. This tool is located in the usr/lpp/WebSphere/bin directory. To use this tool:

1. Define an MVS user ID which can be used to execute the JSP precompile script. This user ID should be defined with an OMVS segment which specifies the same UID and GID as the user ID used to start the server regions in which the compiled JSPs will be used.

   Defining a separate user ID under which to run the JspBatchCompiler.sh script provides better audit and control of this function, independent of the user ID which is used to start the server regions. Two or more accounts can share the same OMVS UID.

   The UMASK value for this user ID must match the value specified for the J2EE server that is going to reference the pre-compiled JSPs. To set this value issue the following command:

   ```
   export _EDC_UMASK_DFLT=xxx
   ```

   where xxx is the umask value to use.

   See "Steps for creating JCL procedures for the control and server regions" on page 146 for a description of how the UMASK value is established for a J2EE server.

2. Add the `com.ibm.ws390.install.root=`*`install_root`* property to the jvm.properties file, if is not already there. If it is already there, make sure *install_root* specifies the fully qualified directory path and file name for the HFS on which WebSphere for z/OS is installed.

   Note: If you want additional debugging and tracing information, such as classpath, time to compile the source, and the version of the compiler, generated during the compile, set the **service.debug.enabled** property in the jvm.properties file to true. The resulting debugging and tracing information will be added to the log file.

3. At an OMVS command prompt, enter the following commands to make sure JAVA_HOME, WAS_HOME and WAS_CONFIG_ROOT are set to the correct values:
   ```
   echo $JAVA_HOME
   echo $WAS_HOME
   echo $WAS_CONFIG_ROOT
   ```
   JAVA_HOME must be set to the exact location of the required level of the Software Development Kit (SDK) on your system. If it is not set to the correct location, enter the following command to change its value:
   ```
   export JAVA_HOME=SDK_install_root
   ```
   WAS_HOME must be set to your WebSphere for z/OS *install-root*. If WAS_HOME is not set to the correct value, enter the following command to change its value:
   ```
   export WAS_HOME=install_root
   ```
   WAS_CONFIG_ROOT must be set to the root directory for your WebSphere for z/OS Application Server configuration files. If WAS_CONFIG_ROOT is not set to the correct value, enter the following command to change its value:
   ```
   export WAS_CONFIG_ROOT=configuration_root
   ```

4. Enter the following command, on a single line, at an OMVS command prompt to set the DB2LIB variable to the classpath for the DB2 class files:
   ```
   export DB2LIB=db2_install_directory/classes/db2j2classes.zip:
      db2_install_directory/classes/db2jdbcclasses.zip:
      db2_install_directory/classes/db2sqljclasses.zip
   ```
   db2_install_directory is the directory where DB2 is installed on your z/OS or OS/390 system.
   Optionally, these DB2 class files can be added to the CLASSPATH environment variable.

5. Enter the following command on a single line at an OMVS command prompt to invoke the JspBatchCompiler.sh script:
   ```
   JspBatchCompiler.sh -node.name <sysplex_name>
           -server.name <server_name>:<server_instance_name>
           [-enterpriseapp.name <name>]
           [-webmodule.name <name>]
           [-filename <jsp name>]
       [-keepgenerated <true|false>]
   ```
   where:

   node.name
      Specifies the name of the sysplex in which the application is deployed. A value **MUST** be specified for this parameter.

server.name
Specifies the name of the J2EE server and J2EE server instance in which the application is deployed, (i.e, BBS1121:BBS1121A). A value **MUST** be specified for this parameter.

enterpriseapp.name
Specifies the name of the enterprise application you want to compile. If this parameter is omitted, all enterprise applications residing on the J2EE server instance specified on the `server.name` parameter will be compiled.

webmodule.name
Is the name of the specific WAR file that you want to compile. If this parameter is omitted, all WAR files for the enterprise application specified on the `enterpriseapp.name` parameter will be compiled.

filename
Is the name of a single JSP file that you want to compile. If this argument is not set, all files in the WAR file specified on the `webmodule.name` parameter will be compiled.

keepgenerated
Is the option to save or erase the generated files. If set to true, WebSphere Application Server saves the generated .java files used for compilation on your server. By default, this is set to false and the .java files are erased after the class files have compiled.

Notes:

a. If you include a parameter, you MUST specify a value for that parameter.
b. If the names specified for these arguments are comprised of two or more words separated by spaces, you must add quotation marks around the names.

Steps for configuring trust association

To provide a trust association interceptor (TAI) implementation for use by WebSphere for z/OS, you must perform the following steps:

1. Ensure that your WebSphere for z/OS system is at the appropriate service level. Trust association interceptors support is provided in APAR PQ55181, which is included in PTF UQ90049, Service Level 11. This PTF must be installed on your WebSphere for z/OS system in order for you to use trust association interceptors with WebSphere for z/OS.

2. Add the following property to your J2EE server's jvm.properties file to enable Version 2 of the Web Security Collaborator:
```
WEB_SECURITY_VERSION=2
```
Version 2 of the Web Security Collaborator contains support for utilizing trust association interceptors as part of the authentication process. If this property is not included in the jvm.properties file or a value other than 2 is specified, WebSphere for z/OS will not perform trust association interceptor processing.

3. Add the following environment variable to your J2EE server's current.env file:
```
 ENABLE_TRUSTED_APPLICATIONS=1
```
This variable Indicates that applications loaded in this address space are to be trusted.

In order to perform trust interceptor processing, WebSphere for z/OS makes use of native APIs that allow for establishing a security context without providing authentication data, such as a password. These are non-public, guarded interfaces, and you must indicate to WebSphere for z/OS that you trust these applications not to be malicious and not to attempt to discover or make use of these native APIs.
If this variable is not set to 1, WebSphere for z/OS will not make use of the trust association interceptors during application processing.

---

4. Add the following properties to the WebSphere for z/OS webcontainer.conf configuration file:

   ```
   WebAuth.TrustAssociationInterceptor.<value>.ImplClass=<classname>
   ```
   This property enables the trust association interceptor support.

   classname
   is the fully qualified name of the class containing the trust association interceptor implementation

   value
   is a unique string of alphanumeric characters that is used to correlate a TAI with its property file. Even if a property file is not being using, a character string, such as TA1, must be included as a place holder.

   , and <filename> is the name of the properties file for that TAI.

   ```
   WebAuth.TrustAssociationInterceptor.<value>.Properties=<filename>
   ```
   This property is only required if you are using a TAI properties file to provide property settings to the trust association interceptor implementation class.

   filename
   is the name of the trust association interceptor properties file.

   value
   is a unique string of alphanumeric characters that is used to correlate this property file to a TAI. This value must match the value specified on a **WebAuth.TrustAssociationInterceptor.**<value>**.ImplClass** property in the webcontainer.conf file.

   Note: If you are using multiple TAIs, you must add a **WebAuth.TrustAssociationInterceptor.**<value>**.ImplClass** property and, optionally, a **WebAuth.TrustAssociationInterceptor.**<value>**.Properties** property to the webcontainer.conf file for each TAI you are using. <value>must be a unique string for each TAI. For example, if you have three TAIs, two that use a properties file and one that doesn't, you might add the following five properties to the webcontainer.conf file:

   ```
   WebAuth.TrustAssociationInterceptor.TA1.ImplClass=class1>
   WebAuth.TrustAssociationInterceptor.TA1.Properties=ta1props
   WebAuth.TrustAssociationInterceptor.TA2.ImplClass=class2
   WebAuth.TrustAssociationInterceptor.TA2.Properties=ta2props
   WebAuth.TrustAssociationInterceptor.TA3.ImplClass=class3
   ```

   These properties are described in more detail in Appendix B, "Default webcontainer.conf file," on page 351.

Steps for enabling dynamic fragment caching

To enable dynamic fragment caching with XML:

1. Create a global configuration file called dynacache.xml file.
   a. In the /usr/lpp/WebSphere/bin directory, locate the dynacache.sample.xml file. This sample file is a valid cache configuration file that enables and configures dynamic fragment caching (also known as servlet caching) with default values.
   b. Make a copy of the dynacache.sample.xml file, name it dynacache.xml, and place it in the directory specified on the `com.ibm.ws390.wc.config.dynxmlfilename` JVM property.

      Note: If you are migrating a dynacache.xml file from a WebSphere Application Server for Distributed Platforms environment, instead of making a copy of the dynacache.sample.xml file provided with WebSphere for z/OS, you can use a copy of the dynacache.xml file you used in the Distributed Platforms environment.
   c. Configure the overall operation of the cache, such as its size, and register the external caches that are used by the application.

      Note: The DTD for the dynacache.xml file is specified in the dynacache.dtd file, which is located in the /usr/lpp/WebSphere/dtd directory. The dyncache.dtd file defines the root element, <cacheUnit>, and should be included in the dynacache.xml file through the DOCTYPE declaration. The beginning of the dynacache.xml must include the following processing instructions:

      ```
      <?xml version="1.0" encoding="UTF-8" ?>
      <!DOCTYPE cacheUnit SYSTEM "dynacache.dtd">
      ```

   If caching is enabled, but a dynacache.xml file is not found in the directory you specified on the `com.ibm.ws390.wc.config.dynxmlfilename` JVM property, the Web container processes servlets in the usual manner. If this file is found, dynamic fragment caching is enabled.

   Note: The dynacache.xml file is read only once, at WebSphere for z/OS startup. If changes are made to the file after startup, WebSphere for z/OS must be stopped and started again before the changes will take effect.

---

2. Create a servlet configuration file.
   a. In the /usr/lpp/WebSphere/bin directory, locate the servletcache.sample.xml file. This sample file includes the servlet definition for WebSphere for z/OS's default server's Snoop servlet.
   b. Make a copy of the servletcache.sample.xml file, rename it servletcache.xml, and place it in the directory you specified on the `com.ibm.ws390.wc.config.dynsrvxmlfilename` JVM property .

      Note: If you are migrating a servletcache.xml file from a WebSphere for Distributed Platforms environment, instead of making a copy of the dynacache.sample.xml file provided with WebSphere for z/OS, you can use a copy of the servletcache.xml file you used in the

Distributed Platforms environment. Just place it in the directory you specified on the `com.ibm.ws390.wc.config.dynsrvxmlfilename` JVM property.

With caching enabled, WebSphere for z/OS will look in the directory specified on the `com.ibm.ws390.wc.config.dynsrvxmlfilename` JVM property for the servlet configuration file that declares which servlets should be cached.

---

3. Add the following properties to the jvm.properties file:
   - com.ibm.ws390.wc.config.dynxmlfilename=*path*.dynacache.xml
   - com.ibm.ws390.wc.config.dynsrvxmlfilename=*path*/servletcache.xml

---

4. Stop and restart WebSphere for z/OS.

---

5. Install the WebSphereSampleApp.ear in the directory /usr/lpp/WebSphere/samples if it is not already installed.

---

6. Verify the cacheable page.

   Using a Web browser, access the following URI to view the Snoop servlet in the default application:

   ```
   /servlet/snoop
   ```
   Invoking and reloading the URI several times should return the same output for the Snoop servlet. If this page is not being cached, the Snoop servlet will display different information every time it is requested.

---

7. Update the servletcache.xml file.

   The root element of the servletcache.xml file is <servletCache> and contains a <servlet> element for each servlet that is going to be dynamically cached. Within the <servlet> element, you must specify parameters that:
   - Identify the servlets to be cached.

     The cache will parse the servletcache.xml file on startup, and extract from each <servlet> element a set of configuration parameters. Then, every time a new servlet is initialized, the cache attempts to match that servlet to a servlet element in this file to determine the configuration information for that servlet. You can specify which servlets to cache in two ways:
     - By specifying the class name of the servlet, or
     - By using the servlet's full URI, beginning after the host name/IP address.

     To cache a servlet class, use the <servletimpl> tag:
     ```
     <servletimpl class="ClassName" />
     ```
     where ClassName is the name of the class to be cached. This class must extend HttpServlet. Whenever a servlet of the specified class is initialized, the dynamic caching function will match that servlet with the element configuration.

     Note: This comparison is done by matching strings; subclasses of the specified servlet implementation will not match this declaration.

     To specify a Web path for your servlets, use the <path> tag.
     ```
     <path uri="web_path" />
     ```

where web_path is the full URI of the servlet, from the hostname to the CGI variable, but not including the CGI variable. The Web application's Web path must be a part of this parameter. So if you access a cacheable servlet with the URL
http://servername.ibm.com/webappname/servlet/MyServlet?arg1=1&....
then that servlet's path element should read:

```
path uri="/webappname/servlet/MyServlet" /
```

You can specify different path elements referring to the same servlet. Also, the same path can appear in two different <servlet> elements, but when configuring a servlet invoked from that path, dynacache will use the configuration from the first valid <servlet> element that matches. For example, if your servletcache.xml file contained the following tags:

```
<servlet>
   <servletImpl class="CalcServlet" />
     :
(other config info)
     :
</servlet>
```

WebSphere for z/OS will cache every servlet of class ″CalcServlet″ across the entire server, regardless of the URI. This is referred to as a general configuration. It is limited in that if there is a class, such as ″ScientificCalculatorServlet″, that extends ″CalcServlet″, ″ScientificCalculatorServlets″ will not match the ″CalcServlet″ element.
If your servletcache.xml file contained the following tags:

```
<servlet>
   <path uri="/tools/Calc" />
   <path uri="/tools/servlet/CalcServlet" />
   <path uri="/tools/Calculator.jsp" />
        :
(other config info)
     :
</servlet>
```

The results would be slightly different. This is an example of a specific setup, which will cache the defined servlet Calc (which may or may not be of the CalcServlet class), and will only cache CalcServlets if they are specifically invoked. In this example, if this servlet was running within a hypothetical tools Web application:

– The first path will catch any request for the /tools/Calc URI, regardless of that servlet's class
– The second path will catch any calls to the Invoker Servlet for the CalcServlet class.
– The third path allows you to cache on a JSP implementation of the calculator.

• Govern the creation of entry IDs for servlets.

  After you declare which servlets to cache, you must define how to cache each servlet. To define how to cache each servlet, you must:
  – Build unique entries for different requests, and
  – Remove these entries at the appropriate time.

  You can do this using the servletcache.xml file, or by writing your own class to handle invalidation. The cache removes entries in the following circumstances:
  a. One of the cache's invalidation methods (see com.ibm.websphere.servlet.cache.Cache in Javadoc) was called directly,

inside the servlet code. This case is not relevant in this context, as it is done programmatically inside an application.

b. Running a servlet triggered a rule based invalidation. This case results from the ″invalidate″ attribute used with a cache variable

c. The timeout for the entry expired or the cache was full and a new entry replaced an old These cases are configured using the <timeout> and <priority>tags:

```
<timeout seconds="time_in_cache" />
```

time_in_cache is the length of time, in seconds, after creation of an entry, that it should be removed from the cache. This value is required. If this value is zero or negative, the entry will not timeout, and can only be removed when the cache is full or programmatically from within an application. If the <timeout>element is not present, then the <servlet> element in which it is contained will be invalidated, and will not be cached. When entries with a positive timeout are created, the timeout is added to the current time to determine when the entry will be invalidated. When that time is reached, if the entry remains in the cache, the cache will force its invalidation.

```
<priority val="priority" />
```

priority is either zero or a positive integer. It is used to initialize a ″current priority″ when a cache entry is created, and every time that that entry is referenced. If this element is not present, then the default value, defined in the dynacache.xml file, will be used.

Servlets are initially loaded into a group with other servlets that have the same ″current priority″. When the cache is full, the LRU algorithm will initially only be applied to the priority group with a value of zero. If there are no entries in this group, the ″current priority″ of all groups is decremented by one until there is a group with a priority of zero that has at least one entry that can be invalidated. Therefore, a servlet that is given a higher priority, or that is frequently requested, is more likely to stay in the cache than one with a lower priority, or that is infrequently requested.

Note: Because the length of time specified on the <timeout> tag applies to **all** entries in the cache, (whether or not the cache is full) servlets will be removed from the cache when their timeout expires even if they have a high priority or are frequently requested.

In a high volume application where space in the cache is at a premium, designers should consider increasing the priority of a servlet or JSP if calculating its output is significantly harder than average, or if it will be executed much more often than average.

Note: Priority values should be kept low, as higher values will not yield a relative improvement, but will use extra LRU cycles. Declaring a servlet with priority 3 and another with priority 4 will generate the same invalidation behavior as servlets with priority 1 and 2, only marginally slower. When dealing with caches in the thousands of entries and greater, that slowdown becomes significant.

Use the timeout variable to guarantee the validity of an entry. Use priority to rank the relative importance of that entry. Giving all elements equal priority results in a standard LRU cache that increases performance significantly, but you can tailor the cache’s operation to the application with careful use of the priority variable.

Other tags used to configure caching are <Externalcache>, <IdGenerator>, and <MetaDataGenerator>:

`<externalcache id="group_name" />`
group_name is the name of an external cache group defined in the global configuration of the cache. When this page is cached, a copy of it will now be pushed to this external cache group. See "External caching" on page 102 for more information about external caching.

`<idgenerator class="class_name" />`
class_name is the full package and class name of a class extending com.ibm.servlet.dynacache.IdGenerator. See "Custom ID and MetaData generators" on page 104 for more information.

`<metadatagenerator class="class_name" />`
class_name is the full package and class name of a class extending com.ibm.servlet.dynacache.MetaDataGenerator. See "Custom ID and MetaData generators" on page 104 for more information.

- Specify how to cache

  Each time a servlet is called and WebSphere for z/OS generates a corresponding HttpServletRequest object, the cache uses information in that object to build an ID string to represent the call. A servlet's cache policy contains the rules that determine which pieces of information are used in that ID string. It always contains certain default information, such as the URI of the requested fragment and its character encoding. The rules give extra information about what variables should be used in the ID, and how they should be treated.

Just like the dynacache.xml file, the servletcache.xml file is read only once, when WebSphere for z/OS is first started. So if changes are made to the file, WebSphere for z/OS must be restarted before the changes will take effect.

The DTD for the servletcache.xml file is specified in the servletcache.dtd file, which is located in directory /usr/lpp/WebSphere/dtd. The servletcache.dtd file defines the root element <servletCache>. It is included in the servletcache.xml file through the DOCTYPE declaration.

The beginning of the servletcache.xml should have the following processing instructions:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE servletCache SYSTEM "servletcache.dtd">
```

---

8. Set up the Dynamic fragment caching monitor .

   WebSphere for z/OS provides the Servlet Cache Monitor application for inspecting the contents and behavior of the fragment cache. To set up the Servlet Cache Monitor application:

   a. Install the ServletCacheMonitor.ear file on each WebSphere for z/OS J2EE server that uses the dynamic fragment cache function. This file is located in the /usr/lpp/WebSphere/samples directory. This EAR file can then be accessed by each J2EE server instance associated with any of these J2EE servers.

      Note: Remember that only one server region can be defined for any server instance that you want to monitor.

   b. Configure the HTTP Transport Handler to monitor different WebSphere for z/OS J2EE server instances in the same sysplex, by specifying a different port for each J2EE server instance. This allows you to use different URIs to access the different server instances' cache monitors.

Note: Remember that only one server region can be defined for any of these server instances.

c. Access the Servlet Cache Monitor through the /servletcache URIs for the J2EE server instances you want to monitor.

Using cache variables

A cache variable is a generic term for a variable whose data should be used in caching a fragment. The four types of cache variables that correspond to the main sources of input for a fragment are:

- request parameters
- request attributes
- session attributes
- cookies

A client browser can set request parameters with CGI when submitting a form, while a servlet can set attributes on an HttpServletRequest object, then forward or include that request to another servlet. WebSphere for z/OS can maintain session attributes that a servlet might want to access, and set cookies for later servlets to process.

In addition to building cache IDs, these variables are used to control the following:

- Grouping of cache entries
- The invalidation of other groups
- Determining whether to cache a servlet based on that variable's value

The cache can decide whether or not to cache an invocation depending on the value of its input variables using the <exclude> tag. Groups are handled by data IDs, which consist of strings specified in the cache variable combined with the variable's value. Using the data_id attribute builds a group name and adds a servlet's cache entries to that group. The invalidate attribute causes the eviction of a group from the cache. It is possible to have an entry that does no caching, and only invalidates groups. The <invalidateonly> tag is provided to save processing time for this case.

The syntax for the various attributes follow:

```
<request>
   <parameter id="parameter_name"
         data_id="group_identifier"
         invalidate="group_identifier"
         ignorevalue="true|false"
         required="true|false">
      <exclude value="exclude_value"/>
   </parameter>
   <attribute id="attribute_name"
         method="method_name"
         data_id="group_identifier"
         invalidate="group_identifier"
         ignorevalue="true|false"
         required="true|false" >
      <exclude value="exclude_value"/>
   </attribute>
</request>
```

where:

parameter_name
Specifies the name of a request parameter, the value of which will be used to generate cache entry IDs.

attribute_name
Specifies the name of a request attribute. The output of one or more of this object's methods will be used to generate cache entry IDs.

method
Specifies the name of a method in this request attribute. The output of this method will be used to generate cache entry IDs. If a method is specified, parenthesis should not be included in the method name. If no method is specified, ″toString″ is assumed.

group_identifier
Is a string that, when combined with the value of this request variable, generates a group name for this cache entry. If used by a ″data_id″ attribute, this cache entry will be placed in that group. If used in an ″invalidate″ entry, then that group will be invalidated, and this entry will then be placed in the cache. If this variable is not present, the group ID will not be used, and neither action will occur.

ignorevalue
Indicates whether this variable's value is relevant to the cache ID, or whether only the variable's presence is important. If true, the cache ID will reflect that the variable was present, but its value will not be used. This value defaults to false when not set.

required
Indicates whether this value must be present in the request. If this is set to true and either the parameter/attribute is not present or (when defining an attribute) does not contain the specified method, then this request will not be cached. This value defaults to false when not set.

You can define unlimited parameter and attribute objects within a <request> element, but you should only define one <request> element per <servlet> element. The cache's handling of session attributes is identical to its handling of request attributes.

```
<session id="session_attribute_name"
         method="method_name"
         data_id="group_identifier"
         invalidate="group_identifier"
         ignorevalue="true|false"
         required="true|false" >
      <exclude value="exclude_value"/>
</session>
```

where:

session_attribute_name
Is the name of a session attribute, the value of which will be used to generate cache entry IDs.

method
Is the name of a method in this session attribute. The output of this method is used to generate cache entry IDs. If this value is not specified ″toString″ is assumed. This method may not take any arguments, and parenthesis should not be included in the method.

group_identifier
: Is a string that, when combined with the value of this request variable, generates a group name for this cache entry. If used by a ″data_id″ attribute, this cache entry will be placed in that group. If used in an ″invalidate″ entry, then that group will be invalidated, and this entry will then be placed in the cache. If this variable is not present, the group ID will not be used, and neither action will occur.

ignorevalue
: Indicates whether this variable's value is relevant to the cache id, or if only the variable's presence is important. If true, the cache id will reflect that the variable was present, but its value will not be used. This value defaults to false when not set.

required
: Indicates whether this value must be present in the session. If this is set to true and either the session parameter is not present or does not contain the specified method, then this request is not cached. This value defaults to false when not set.

As with request parameters/attributes, there can be any number of session attributes declared in a servlet.

Notes:

1. If the <invalidateonly> tag is used, no caching is performed for this servlet, though invalidations triggered by it will take effect.
2. If you attach an <exclude> tag to a cache variable (<exclude_value>), it will keep a servlet from being cached when that variable is present on the request. The <exclude> tag can only be applied for certain values, or for all values of a cache variable. When the exclude tag is used without its ″value″ attribute, no caching is performed for this servlet when the variable is present. Alternatively, users can repeat the <exclude> tag with the ″value″ attribute defined several times to specify a set of values that keep the servlet from being cached. This helps prevent caching of control servlets.

For example, the following setup indicates that whenever the request parameter ″nocache″ is present, the servlet should not be cached:

```
<servlet>
   <path uri="/myServlet"/>
   <timeout seconds="-1" />
   <request>
      <parameter id="nocache" >
            <exclude/>
      </parameter>
   </request>
</servlet>
```

In this example, the servlet will be cached unless the request parameter ″cache″ is present and its value equals ″no″ or ″false″:

```
<servlet>
   <path uri="/myServlet" />
   <timeout seconds="-1" />
   <request>
      <parameter id="cache" >
            <exclude value="no"/>
            <exclude value="false"/>
      </parameter>
   </request>
</servlet>
```

Exclude tags can be applied to any cache variable.

Notes:

1. Invalidating has a higher performance cost than caching. Avoid using the same variable to invalidate a group and to group an entry. You will see less performance benefit than when just using the variable to define a group id.
2. Data_id and invalidate tags that are within the same <servlet> element should have different values, usually. If they have the same value, the group is invalidated, and the entry for the current servlet is put into that group. The invalidate tag that corresponds to a data_id will occur in different <servlet> elements.

Summary of the elements in a servletcache.xml file

Following is a summary of the elements in a servletcache.xml file:

```
1.  <?xml version="1.0" encoding="UTF-8" ?>
2.  <!DOCTYPE cacheUnit SYSTEM "dynacache.dtd">
3.  <servletCache>
4,  <servlet>
5.  <invalidateonly/>
    <servletimpl class="some.package.SomeClass"/> OR
            <path uri="MyServlet" /><path uri="servlet/SomeClass"/> OR
            <metadatagenerator class="package.GeneratorClass" />
6.  <timeout seconds= "timeout value" />
    <priority value="priority value" />
7.  <externalcache id="external cache group name">
8.   OR
9.  <request>
10. <parameter id="parameter name"
           data_id="group identifier 1"
       invalidate="group identifier 2"
       ignorevalue="true|false"
       required="true|false">
    <exclude value="exclude value"/>
    </parameter>
11. <attribute id ="attribute name"
        method= "aMethodName"
        data_id="group identifier 1"
        invalidate="group identifier 2"
        ignorevalue="true|false"
        required="true|false" />
12. </request>
13. <session id="session attribute name"
        method="aMethodName"
        data_id="group identifier 1"
        invalidate="group identifier 2"
        ignorevalue="true|false"
        required="true|false" />
14. <cookie id="cookie name"
        method="aMethodName"
        data_id="group identifier 1"
        invalidate="group identifier 2"
        ignorevalue="true|false"
        required="true|false" />
15. <idgenerator class="package.IdGeneratorClass" />
16. <metadatagenerator class="package.MetaDataGeneratorClass"/>
17. </servlet>
18. </servletCache>
```

Notes:

1. Line 6: Timeout <1 implies the value will not time out. Required.
2. Line 8: OR applies to lines 10-15

3. Line 10: The method, called from the request parameter, defaults to toString, and required defaults to false.
4. Line 10: The exclude tag can be used in any of the four cache variable types.
5. Line 10: The method called on the request attribute defaults to toString, and required defaults to false.
6. Line 13: The method called on the session attribute defaults to toString, and required defaults to false.
7. Line 14: The method called on the cookie defaults to toString, and required defaults to false.

Dynamic fragment cache XML examples

In this example, Calculator Servlet is defined inside a ′tools′ Web application. It takes an operation ′operation′ and two arguments, ′arg1′ and ′arg2′. The values for these variables are received in the query string from an external user, that is, request parameters from a browser or applet. You invoke the servlet to calculate 2+3 to get the answer 5, with the URI:

```
/tools/Calc?arg1="2"&arg2="3"&operation="+"
```

To cache the output of this servlet, you distinguish results using the request parameters, so that 2,3,+ has a different cache entry than 4,5,*. For this example, you would define the following <servlet> element:

- <servlet>
- <path uri=″/tools/Calc″ />
- <request>
- <parameter id=″arg1″ required=″true″ />
- <parameter id=″arg2″ required=″true″ />
- <parameter id=″operation″ required=″true″ />
- </request>
- <timeout seconds=″-1″ />
- </servlet>

In a second example, the news servlet of class CoastalNewsServlet displays either west coast news or east coast news, depending on a user's location. This servlet has a session object named ′location′ of class ′LocationBean′ with a method getCoast() that returns ″east″ or ″west″. If this object is not present on the session, then the servlet returns the news for both coasts, which you also want to cache. So whether or not the location is present on the session, you want to cache the output of the servlet, and you do not want the entries to time out:

```
<servlet>
<servletimpl class="CoastalNewsServlet" />
<session id="location" method="getCoast" />
<timeout seconds="-1" />
</servlet>
```

To group cache entries based on the coast, or to invalidate the cache entries for a coast whenever the location bean gets updated by a control servlet, you must consider the design of the application. If a variable is not available to a servlet at execution (that is, the request/session variable has not been set), then, even if the servlet is cacheable, no group id will be generated based on that variable. In the CoastalNewsServlet example, a missing session parameter causes the servlet to display the news for both coasts. Naturally, in this case, you want the cache entry to belong to both the east and west coast groups. However, because the cache does not handle data ids when variables are missing, this is not possible. The simplest solution to this problem marks the location bean as a required variable for caching. This means the output of the news servlet will not be cached if location is undefined. Therefore, you can now do all the grouping knowing that the location bean will be present to place entries into the correct groups. Servletcache.xml needs two sets of changes to finish configuring groups:

1. The CoastalNewsServlet entry must be modified to put entries into groups, and
2. A new entry for the control servlet that updates the location bean must be added to allow invalidation of these groups.

```
<servlet>
<servletimpl class="CoastalNewsServlet" />
<session id="location" method="getCoast" data_id="coast" required="true" />
<timeout seconds="-1" />
</servlet>
<servlet>
<invalidateonly/>
<servletimpl class="LocationUpdateServlet" />
<request>
<attribute id="new_coast" invalidate="coast" />
</request>
</servlet>
```

Now, when the news servlet is invoked, the cache will take the data_id ″coast″ and append ″=″ and the value of location.getCoast() to create a group name to identify that entry. In the update servlet, a new_coast string is expected as a request attribute, and will be used in the same fashion to build group names for removal from the cache.

Using HttpSession and request attributes

The dynamic cache can use objects stored in an HttpSession or request attribute in caching requests. The Servlet specification allows the Servlet or JSP programmer to put any number of objects into the session or request, and index these values with a String key name. Possible uses range from the storage of minimal data and simple types (for example, String, Integer, Boolean) to very complex and large (Vectors and Hashtables of complex User Objects). Given this background, this is what the WebSphere for z/OS Servlet/JSP cache supports:

- When building cache policies the user can specify: A key name for a session/request attribute value and
- An [optional] method name used for retrieving a value from the stored object.

In the case where the method name is not specified, the default method toString() is used to transform the object into a String for use in the cache ID.

Consider the following object:

```
class User {
public String getName() {...};
public String getPrimaryGroup() {...};
 }
```

If an instance of this object is stored in the session using the key ″user″, you might specify the following cache policy: (in this example, imagine that different user groups view a different home page):

```
<servlet>
<path uri="/index.jsp" />
<session id="user" method="getPrimaryGroup" />
<servlet>
```

Caching personalized pages

If you want a fragment of a personalized page, such as a stock list owned by an user, the fragment would have two servlets:

1. The first obtains the stock list from the database and forwards the list of stock symbols to the second servlet
2. The second servlet gets quotes for the stocks from the back end.

Depending on exactly how this stock list is generated, caching will have varying effectiveness, but it will provide a benefit. When determining how to cache this page, you need to apply a key concept in servlet caching. The biggest performance wins come from caching servlets that obtain information from outside WebSphere for z/OS. This means that while caching a simple presentation JSP file will give moderate performance gains, caching servlets that request information from Enterprise Java Beans or databases, saves WebSphere for z/OS processing power and decreases load on the back end.

In this example, both sets of actions can be cached (since they are both reads from the database). The first servlet is a classic example of a cacheable servlet. It is a simple database lookup, using one value as input (a user ID), and producing the list as output. The interesting case (an example of designing an application with caching in mind) is how the servlets go from a list of symbols to actually looking up the quotes.

The simple design involves one servlet that does all the work and a presentation JSP. The servlet gets a list of stock symbols for a user from the database (presumably that user’s database record holds the symbols in their portfolio), then immediately goes back to the database and looks up current quotes for those symbols. It builds a list of quotes and forwards them to a presentation JSP.

This design can be cached. You base everything off the user ID, and cache the current quotes for that user. There are missed opportunities here, though. First, since everything is based off of an individual user, you cannot reuse the stock quotes gathered for other users who own the same stocks. The usefulness of this cache entry is limited to one user. Second, whenever one of the quotes, or the list of symbols changes, the whole page changes. This entry is unstable, and will not be correct very long.

A better design fragments the data requests into 3 different servlets/JSP files, separating each of the two database reads into its own servlet/JSP file. The first servlet would use the user id to get the list of symbols. It would forward the list to a presentation JSP that would format the list, and get individual quotes from a third servlet that takes a symbol as input.

All three of these servlets/JSP files can be cached individually. The first servlet is caching the user specific list of symbols only. Changes to a user’s portfolio will not affect the cache entries for the stock quotes. The presentation JSP file only changes the list of symbols supplied to it. Caching in this example is as useful as caching the first servlet, and could be even more useful if different users have the same list of stocks. The quotes are cached with the last servlet, which will have a different entry for each stock. If one stock quote changes, then the request for that individual stock will have to be reinvoked from the application server. The rest of the requests can all be served from the cache. Most importantly, these quotes are reusable in any request for a customer’s portfolio. By narrowing the responsibility of individual pieces of the application, you can provide selective caching, and achieve bigger performance gains.

Building a custom ID generator

You can build your own ID generator, but it must implement the com.ibm.websphere.servlet.cache.IdGenerator interface. There are three methods in this interface:

- public void initialize(CacheConfig cc);
- public String getId(ServletCacheRequest request);
- public int getSharingPolicy(ServletCacheRequest request);

The initialize method is called during startup. Normally, the cache processes a servlet's XML configuration and builds a CacheConfig object that is made available to the IdGenerator. The initialize method then builds a list of request and session variables that must be included in the cache ids for the servlet. Since the ″plugged-in″ IdGenerator is created with a specific servlet's behavior in mind, working with the CacheConfig is unnecessary; just hard code the configuration into the getId method.

The getId method returns the unique String cache id when the servlet is invoked. If the servlet is cached, the getId method returns null. Typically, an Id will incorporate the following:

- The URI of the servlet
- The character encoding of the request (when the result is not null)
- The names and values of the input variables that determine the servlet's output

The getSharingPolicy method should return EntryInfo.NOT_SHARED.

Chapter 9. Creating and running WebSphere for z/OS client applications

Once you have installed application components in a J2EE server, you are ready to create WebSphere for z/OS client applications that use those components. Figure 1 on page 4 illustrates the most likely types of client applications for components that run in a J2EE server:

- Application clients that run only on non-z/OS or non-OS/390 platforms, such as Windows 2000. These application clients are specific module types defined in the Sun Microsystems J2EE specification:
  - J2EE application clients, which access Enterprise beans, JDBC databases and other resources, and depend on an application client run-time to configure its execution environment.
  - Java thin application clients, which use a light-weight Java client programming model. This type of client is best suited for use in situations where a Java client application exists but the application must be enhanced to make use of Enterprise beans, or where the client application requires a thinner, more light-weight environment than the one offered by the J2EE application client.
- RMI clients that run on z/OS or OS/390. These Java programs have full access to J2EE application components installed in WebSphere for z/OS J2EE servers, but lack access to the java:comp JNDI namespace.

The following table lists types of client applications, and where to find information about creating and running them.

For this type of client:	See the following information sources:
Application clients running on non-z/OS or non-OS/390 platforms	"Application clients that run on non-z/OS platforms"
Clients running in WebSphere Standard Edition on z/OS or OS/390	"Java clients running in WebSphere Application Server Standard Edition for z/OS or OS/390" on page 220
Clients running on z/OS or OS/390	"Native z/OS or OS/390 Java clients" on page 222

Application clients that run on non-z/OS platforms

Application clients that run on non-z/OS and non-OS/390 platforms can be the following specific module types defined in the Sun Microsystems J2EE specification:

- J2EE application clients
- Java thin application clients

Although these clients cannot run on the z/OS or OS/390 platform, they can access J2EE application components (Enterprise beans or Web applications) that run in a WebSphere for z/OS J2EE server.

Depending on the platform on which you plan to run J2EE application clients, you need the following software:

© Copyright IBM Corp. 2000, 2003

If your application clients will run on this type of workstation:	Then you need this additional software:
Windows NT or Windows 2000	You need to download one of the following z/OS packages: • WebSphere J2EE Client for NT • WebSphere Java Technology Client See "Steps for running application clients on Windows NT or Windows 2000" for instructions.
Any other workstation platform supported by WebSphere Application Server, such as AIX and Solaris	You must deploy the J2EE application clients in the WebSphere Application Server Advanced Edition Version 4.0 run-time environment. See the **InfoCenter** information for installing and using WebSphere Application Server Advanced Edition Version 4.0. The **InfoCenter** is available at `http://www.ibm.com/software/webservers/appserv/`

Rules:
- Clients must include the following on their CLASSPATH: the location and name of JAR files for any Enterprise bean that the client uses
- To access the WebSphere for z/OS naming service, clients must set the property:
  ```
  java.naming.provider.url="iiop://x.x.x.x:ppp"
  ```
  where *x.x.x.x:ppp* is the IP address and port of the WebSphere for z/OS systems management server.

Guidelines:
- To develop application clients, you can use any Java application development tools.
- To assemble application clients, you can use either the WebSphere for z/OS Application Assembly tool or the WebSphere Application Server Advanced Edition Application Assembly tool.
- Make sure you consider security requirements. Review the considerations in "Security" on page 19 and consult with your security administrator.

For further information about designing and coding client applications to run on platforms other than z/OS and OS/390, see the **InfoCenter** information for the WebSphere Application Server Advanced Edition Version 4.0, which is available at `http://www.ibm.com/software/webservers/appserv/`

Steps for running application clients on Windows NT or Windows 2000

Before you begin: You need to:
- Decide whether you are using the WebSphere for z/OS Application Assembly tool or the WebSphere Application Server Advanced Edition Application Assembly tool. If you are using the WebSphere for z/OS Application Assembly tool, download the latest copy. For additional details, see "Steps for installing the Application Assembly tool" on page 136.
- Make sure the J2EE application components that your client uses are installed on the WebSphere for z/OS J2EE server, and that the J2EE server is active.

Perform the following steps to prepare and run an application client on Windows:

1. Install the appropriate client package by downloading the following files from the WebSphere for z/OS HFS into a working directory on your workstation:
   - For J2EE application clients, download the WebSphere J2EE Client for NT (`J2EEClient_NT.zip`). Unzip the file on your workstation.
   - For Java thin application clients, download the WebSphere for z/OS Java Technology Client (`JavaTechnologyClient_setup.exe`)

   The default HFS location for these files is `/usr/lpp/WebSphere/bin/`

   Once the files are on your workstation, run the `setup.exe` file.

   Tip: Always make sure you have the latest service for these WebSphere for z/OS packages by checking the Support Web site `http://www.ibm.com/software/webservers/appserv/support.html`

2. Create Java main, using the Java application development tool of your choice.

   Rules:
   - Clients must include the following on their CLASSPATH: the location and name of JAR files for any Enterprise bean that the client uses
   - To access the WebSphere for z/OS naming service, clients must set the property:

     ```
     java.naming.provider.url="iiop://x.x.x.x:ppp"
     ```

     where *x.x.x.x:ppp* is the IP address and port of the WebSphere for z/OS systems management server.

3. Use a WebSphere application assembly tool to package the application client. For example, using the WebSphere for z/OS Application Assembly tool:
   a. Create an application with the application client module (the Java main).
   b. Define the following:
      - Enterprise bean references (`ejb-ref` elements) and their full JNDI look-up names.
      - Resource references (`res-ref` elements) and their full JNDI look-up names.
      - Any environment variable references (`env-vars`), such as time-out values or SQL query strings.
   c. Export the application in an Enterprise Archive (EAR) file.

4. Use the Application Client Resource Configuration tool to define local resources for your Java main. To start the tool, enter `clientConfig` from a Windows command prompt.

   If you need further details for using this tool to define local resources, see the **InfoCenter** for WebSphere Application Server Advanced Edition Version 4.0, which is available at `http://www.ibm.com/software/webservers/appserv/`

5. To start the application client, enter `launchClient` from a Windows command prompt.

If you need further details for using this tool to start the client, see the **InfoCenter** for WebSphere Application Server Advanced Edition Version 4.0, which is available at `http://www.ibm.com/software/webservers/appserv/`

Java clients running in WebSphere Application Server Standard Edition for z/OS or OS/390

Figure 28 on page 366 depicts possible configurations for WebSphere Application Server Standard Edition V3.5 Web applications. The following information applies only to the case in which Web applications run in the V3.5 Standard Edition environment and drive Enterprise beans that run in a WebSphere for z/OS J2EE server, when both the V3.5 product and the WebSphere for z/OS J2EE server reside on the same z/OS or OS/390 system. For information about installing both products on the same system, see *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834.

Because documentation for developing and running this type of J2EE application client is available in *WebSphere Application Server for OS/390 V3.5 Standard Edition Planning, Installing, and Using*, GC34-4835, the following procedure provides only a summary of the development and installation process, with references to resources with further instructions. Use this procedure as a checklist to make sure you have the correct tools and information sources on hand, as you begin to develop and run these clients.

Before you begin:

- You must have the WebSphere Application Server Standard Edition Version 3.5 for z/OS or OS/390 installed and customized. Because this task is typically performed by system programmers, you might need the help of such experts to complete this task at your installation. If necessary, see one or both of the following information sources:
  - *WebSphere Application Server for z/OS or OS/390 Version 3.5 Standard Edition Program Directory*, which is shipped with the Standard Edition product
  - *WebSphere Application Server for OS/390 V3.5 Standard Edition Planning, Installing, and Using*, GC34–4835, which is available at the following Web site:

    `http://www.ibm.com/software/webservers/appserv/`
- Make sure you consider security requirements. Review the considerations in “Security” on page 19 and consult with your security administrator.

Perform the following steps to create and run clients in WebSphere Application Server Standard Edition:

1. Make sure you have installed the appropriate application development software, and have the associated documentation for those tools on hand. For further details, see Chapter 5, “Setting up the application development environment,” on page 109.

   Recommendations:

   - Use an application development environment such as WebSphere Studio Application Developer, Application Developer Integration Edition, or VisualAge for Java to develop and test servlets and JSPs. These environments enable you to test your application clients without having to install a WebSphere Application Server environment on the workstation.

- Make sure you check the latest edition of the following book for any additional application development tooling considerations: *WebSphere Application Server for OS/390 V3.5 Standard Edition Planning, Installing, and Using*, GC34–4835

---

2. Code and test the servlet and JSP components of your J2EE application.

   Rule: J2EE application clients must set the Java property `java.naming.factory.initial` to `com.ibm.websphere.naming.WsnInitialContextFactory`.

   Recommendation: Make sure you check the latest edition of *WebSphere Application Server for OS/390 V3.5 Standard Edition Planning, Installing, and Using* for any programming model restrictions that might affect the design of your application clients.

---

3. When you are satisfied with the unit test results for these components, transfer the component artifacts (JAR files, and so on) to a working directory in the hierarchical file system (HFS) on z/OS or OS/390.

   Recommendation: Use the following naming convention for your working directory: `/webapp/`*`servlet_or_JSP_name`*`/`

   Note: When you transfer the files from the workstation (an ASCII-based system) to z/OS or OS/390 (an EBCDIC-based system), you must perform the necessary conversions as part of transferring the files.

---

4. To the WebSphere Application Server directives section in the HTTP server configuration file (called `httpd.conf`), add the following Service directive for the `webapp/`*`servlet_or_JSP_name`* relative directory:

   ```
   Service /webapp/servlet_or_JSP_name/*
       /usr/lpp/WebSphere/AppServer/bin/was400plugin.so:service_exit
   ```
   The `httpd.conf` file is usually in the `/etc` directory.

---

5. Configure the WebSphere Application Server by changing settings in the `was.conf` file as follows:

   ```
   appserver.libpath=/usr/lpp/WebSphere/lib
   appserver.classpath=path/ws390crt.jar
   ```

   Note: The `ws390crt.jar` file is shipped with the WebSphere for z/OS product, so you must copy it into the HFS for the Standard Edition V3.5 product. The JAR file is in `/usr/lpp/WebSphere/lib/` (if your installation uses the default path `/usr/lpp/WebSphere` when installing the WebSphere for z/OS product). For the `appserver.classpath` setting, substitute *path* with the Standard Edition V3.5 HFS directory into which you copied the `ws390crt.jar` file.

   The `was.conf` file is usually in the `/usr/lpp/WebSphere/AppServer/properties` directory. *WebSphere Application Server for OS/390 V3.5 Standard Edition Planning, Installing, and Using* describes all of the application server properties that you can set through this file.

---

6. Define your J2EE application clients to the WebSphere Application Server by setting `Web application` properties in the `was.conf` file.

Rule: The following property settings are required for both servlets and JSPs:

- `deployedwebapp.<`*`webapp_name`*`>.classpath=` `/usr/lpp/WebSphere/lib/ws390crt.jar` **plus** the location and name of JAR files for any Enterprise bean that the client uses
- `deployedwebapp.<`*`webapp_name`*`>.java.naming.factory.initial=` `com.ibm.websphere.naming.WsnInitialContextFactory`
- `deployedwebapp.<`*`webapp_name`*`>.javax.naming.provider.url=` `"iiop://`*`x.x.x.x:ppp`* where *x.x.x.x:ppp* is the IP address and port of the WebSphere for z/OS systems management server.

  You must specify a value for this property to access the WebSphere for z/OS naming service on another sysplex, or to access the JNDI on an Advanced Edition WebSphere Application Server running on a workstation platform.

Recommendation: Make sure you check *WebSphere Application Server for OS/390 V3.5 Standard Edition Planning, Installing, and Using* to determine whether you need to set any additional `Web application` properties through the `was.conf` file.

---

7. Stop and restart the HTTP server.

---

8. Access your J2EE application client from a Web browser by setting the location to `http://<`*`host`*`>/webapp/<`*`webapp_name`*`>` Web site, where *<host>* is the application client host system.

Native z/OS or OS/390 Java clients

When your installation's system programmers installed and customized WebSphere for z/OS, they ran sample applications (which are shipped as part of the product) to verify that the installation was successful. You can use the same applications to see how to drive J2EE application components on z/OS or OS/390 from a Java client running on z/OS or OS/390. *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834 contains a procedure for running the WebSphere for z/OS installation verification programs (IVPs). Use this procedure and the associated files as models for running your own z/OS or OS/390 clients, which may include any Java processes running in the UNIX System Services (USS) environment. Pay particular attention to the content and instructions in the `ejbivp.sh` file, which is a USS shell script that runs the sample PolicyClient to drive the Policy Enterprise bean IVP. This shell script is available in the `/samples/PolicyIVP/ejb` subdirectory of the HFS location in which WebSphere for z/OS is installed.

For information about designing and coding Java clients that run on z/OS or OS/390, start with *z/OS UNIX System Services User's Guide*, SA22-7801, and the following rules and guidelines:

- **Rules:**
  - Clients must include the following on their CLASSPATH: `/usr/lpp/WebSphere/lib/ws390crt.jar` **plus** the location and name of JAR files for any Enterprise bean that the client uses
  - Clients must use explicit names for Enterprise beans and homes. In other words, clients cannot use JNDI look-ups for java:comp names.
  - To access the WebSphere for z/OS naming service, clients must set the property:

```
java.naming.provider.url="iiop://x.x.x.x:ppp"
```

where *x.x.x.x:ppp* is the IP address and port of the WebSphere for z/OS systems management server.

You must specify a value for this property to access the WebSphere for z/OS naming service on another sysplex, or to access the JNDI on an Advanced Edition WebSphere Application Server running on a workstation platform.

- **Guideline:** Make sure you consider security requirements. For z/OS or OS/390 clients, your installation might not require any security mechanisms even if you are running these clients in a production system. Review the considerations in "Security" on page 19 and consult with your security administrator.

Part 3. Programming and deployment scenarios for J2EE applications

Each of the following topics illustrate how to exploit some of the J2EE technologies and APIs that WebSphere for z/OS supports. Some of these scenarios span the coding, assembly, and installation tasks that application assemblers, deployers, and system administrators perform in the WebSphere for z/OS environment.

The instructions in these scenarios assume that you are using the WebSphere for z/OS Application Assembly tool and Administration application for application assembly, deployment and installation tasks. You may, however, achieve the same results using other WebSphere-supported application assembly tools. In fact, the preferred method of deploying applications is to use WebSphere Studio Application Developer and Direct Deployment Tool/390fy. If you are using the following two IBM extensions, however, you still need to use the WebSphere for z/OS Application Assembly tool:

- **SyncToOSThread:** See “Synchronizing operating system thread identity to RunAs identity” on page 31 for more information on this extension.
- **Connection Management Policy:** See “Exploiting connection management support” on page 71 for more information on this extension.

If you want to:	Then use this topic as a model procedure:
Code and deploy application components that use Java Naming and Directory Interface calls	Chapter 10, “Using JNDI look-ups,” on page 227
Use declarative and programmatic security mechanisms to set authorization controls for Enterprise beans or their methods	Chapter 11, “Using security roles and RunAs identities with Enterprise beans,” on page 231
Code and deploy application components that use Java Message Services	Chapter 12, “Using the Java Message Service API in J2EE application components,” on page 237
Code and deploy application components that use JavaMail APIs	Chapter 13, “Using the JavaMail API in J2EE application components,” on page 241
Use HTTP session support	“HTTP session support” on page 87
Enable security mechanisms for Web applications	Chapter 14, “Steps for configuring Web security,” on page 247
Develop, deploy, register, and enable your own Web services for use	Chapter 15, “Creating and deploying Web Services,” on page 249
Configure Type 4 JDBC connectors for use by Enterprise beans and servlets	Chapter 16, “Using Type 4 JDBC Connectors with WebSphere for z/OS,” on page 257

© Copyright IBM Corp. 2000, 2003

Chapter 10. Using JNDI look-ups

Subtask	Associated information (See . . .)
Understanding the concepts related to JNDI and the WebSphere for z/OS run-time environment	• "Naming" on page 42 • "Java Naming and Directory Interface™ (JNDI)" on page 43
Correctly using `java:comp` to look up a database resource	"Example: Using the JNDI subcontext to look up a resource"
Setting JNDI caching properties for a J2EE server or client	"Example: Modifying JNDI caching behavior" on page 229

Example: Using the JNDI subcontext to look up a resource

The following sample illustrates how an Enterprise bean, using bean-managed persistence, can use the JNDI subcontext (`java:comp`) to refer to its datasource. The bean is part of the PolicyIVP program that your system programmers use to verify the installation of the WebSphere for z/OS product. In this sample, the BMP bean's datasource is a DB2 database on z/OS or OS/390.

The following topics cover the coding, assembly, and deployment tasks that need to be completed to successfully run the bean in a J2EE server:
- "Steps for preparing the Enterprise bean"
- "Setting up the J2EE server and DB2 datasource" on page 228

Steps for preparing the Enterprise bean

1. Using an appropriate workstation development tool, code the BMP bean implementation like this:

   Example:

   ```
   ⋮
    // obtain the initial JNDI context
    Context initCtx = new InitialContext();
    // perform JNDI lookup to obtain resource manager
    // connection factory
    javax.sql.DataSource ds = (javax.sql.DataSource)
     initCtx.lookup("java:comp/env/jdbc/EJB_IVP_DS");
    // Invoke factory to obtain a connection. The security
    // principal is not given, and therefore
    // it will be configured by the Deployer.
    java.sql.Connection con = ds.getConnection();

   ⋮
   ```

   Result: Exporting this BMP bean generates a deployment descriptor that contains the following resource reference for the datasource:

   ```
   <resource-ref>
   <description>A data source for WAS 390 PolicyIVP BMP
   </description>
   <res-ref-name>jdbc/EJB_IVP_DS</res-ref-name>
   <res-type>javax.sql.DataSource</res-type>
   <res-auth<Container></res-auth>
   </resource-ref>
   ```

Note: Note that the `java:comp/env/` prefix is not included in the resource reference element. Because this JNDI subcontext for the BMP bean is implicit, the EJB container in the J2EE server automatically uses the prefix at run-time.

---

2. Use the Application Assembly tool to package the BMP bean in an Enterprise Archive (EAR) file. If you need further details, see "Steps for assembling a new J2EE application" on page 137.

After the bean is packaged in an EAR file, you are ready to install the BMP bean in a J2EE server that is configured to access a DB2 datasource.

Setting up the J2EE server and DB2 datasource

1. Set up the database resources that the BMP bean uses. Your system programmer or database administrator has probably installed and configured the required DB2 subsystem, and might already have created the databases or tables that your application will use. So the only tasks you might need to do are these:
   - Find out what DB2 subsystem name you need to specify when you configure the J2EE server.
   - Find out what security controls, if any, your installation requires for access to the DB2 subsystem and its resources, and use the appropriate security definitions.
   - Create any database tables that the BMP bean needs to use.

   If necessary, see:
   - "Security" on page 19 for security controls that you might need to use.
   - *DB2 Administration Guide*, SC26-9931 for instructions on creating database tables.

---

2. Use the Administration application to define a J2EE resource. Add this definition under the J2EE Resources folder, specifying a name and type in the properties form.

---

3. Use the Administration application to define a J2EE resource instance that represents the DB2 subsystem on z/OS or OS/390. Add this definition under the J2EE Resources folder, specifying the appropriate values in the properties form.

   For example, the instructions for setting up the DB2 datasource for the PolicyIVP tell your system programmer to enter the following values:
   - J2EE resource instance name.

     Example: BBOASR2_EJB_IVP_RESOURCE_*system*, where *system* is the name of the z/OS or OS/390 system on which the J2EE server will run.
   - Location Name: Supply the the DB2 location name.

---

4. Use the Administration application to install the BMP bean:

a. Choose `Install J2EE Application...` from the Selected menu bar. The Install J2EE Application dialog box appears.

b. In the dialog box, enter the fully qualified path name of the EAR file that contains your BMP bean, and the name of the FTP server for the sysplex in which you want to install the bean.

c. Click OK, and the Administration application displays the application content in the EAR file.

d. For BMP bean, use the following steps to resolve the datasource references:

   1) Expand the `EJB Jars` folder to display the BMP bean. When you select the bean, the Administration application displays information from the deployment descriptor.

   2) Using the Reference and Resource Resolution window, select the J2EE resource you defined earlier in this procedure, to link the DB2 subsystem to the bean's resource reference name.

e. Click OK to start the application installation process.

If you need additional details about installing applications, see "Steps for installing a J2EE application" on page 154.

5. Validate the conversation that you have modified or just created. Message BBON0442I appears in the status bar, indicating that the new conversation is valid.

6. Commit the validated conversation. Message BBON0444I appears in the status bar when the new conversation and J2EE server definition are committed.

When you activate this conversation, the BMP bean that you installed may use the tables in the DB2 database.

Example: Modifying JNDI caching behavior

WebSphere for z/OS uses caching to increase the performance of JNDI lookup operations related to both the global and local namespaces. You may change JNDI caching behavior only for the global namespace, not for `java:comp` lookups.

To change cache behavior in a Java client running on z/OS or OS/390, set the JNDI caching properties described in "Java Naming and Directory Interface™ (JNDI)" on page 43, using one of the following:

- An environment Hashtable in the client code.
- A `jndi.properties` resource file.

  Note: Because a default JNDI properties file is shipped with WebSphere for z/OS, Java clients must ensure that their own JNDI properties file appears in the CLASSPATH before the default file. In other words, the client's properties file must precede any WebSphere for z/OS JAR files that the client uses (for example: `ws390crt.jar`).
- The java command line, using the -D switch.

  Example:

```
java  -Dcom.ibm.websphere.naming.jndicache.cacheobject=cleared
com.ibm.ivj.samples.policy.PolicyClient
```

Recommendation: Set caching properties by passing them as a Hashtable or Properties object to the InitialContext constructor. This way there is less confusion about which contexts implement caching and which do not.

The following code sample illustrates how a J2EE application client, running on z/OS or OS/390, may change the default JNDI caching behavior that WebSphere for z/OS uses. You might want to change the default JNDI caching, for example, when your application component works with two types of object references: Those that are stable, and others that are volatile. The sample illustrates how you can use one context that uses caching to lookup the stable objects, and another context that does not use caching to look up the volatile objects.

```
import java.util.Hashtable;
import javax.naming.InitialContext;
import javax.naming.Context;

/*****
Caching discussed in this section pertains only to the WebSphere for z/OS
initial context factory. Assume the property, java.naming.factory.initial,
is set to com.ibm.websphere.naming.WsnInitialContextFactory
as a java.lang.System property.
*****/

Hashtable env;
Context ctx;

// To clear a cache:

env = new Hashtable();
env.put("com.ibm.websphere.naming.jndicache.cacheobject", "cleared");
ctx = new InitialContext(env);

// To set a cache's maximum cache lifetime to 60 minutes:

env = new Hashtable();
env.put("com.ibm.websphere.naming.jndicache.maxcachelife", "60");
ctx = new InitialContext(env);

// To turn caching off:

env = new Hashtable();
env.put("com.ibm.websphere.naming.jndicache.cacheobject", "none");
ctx = new InitialContext(env);

// To use caching and no caching:

env = new Hashtable();
env.put("com.ibm.websphere.naming.jndicache.cacheobject", "populated");
ctx = new InitialContext(env);
env.put("com.ibm.websphere.naming.jndicache.cacheobject", "none");
Context noCacheCtx = new InitialContext(env);

Object o;

// Use caching to look up home, since the home should rarely change.
o = ctx.lookup("com/mycom/MyEJBHome");
// Narrow, etc. ...

// Do not use cache if data is volatile.
o = noCacheCtx.lookup("com/mycom/VolatileObject");
// ...
```

Chapter 11. Using security roles and RunAs identities with Enterprise beans

WebSphere for z/OS supports programmatic authorization controls that can provide authorization checking on a more granular level than your installation's network and system-level authorization controls. These programmatic authorization controls include:

- Security roles, which are logical representations of types of users, to prevent unauthorized use of individual J2EE application components or their methods.
- Security identities that are specified for the execution of a particular method and any downstream processes, or for the execution of an individual application component. (This type of authorization control is similar to the RunAs support defined in the Sun Microsystem's EJB 2.0 specification.)

The following table shows the subtasks and associated information for using security roles and identities in the WebSphere for z/OS environment:

Subtask	Associated information (See . . .)
Understanding the concepts related to programmatic authorization controls	"Authorization controls for J2EE application components" on page 25
Completing coding, assembly, and deployment tasks to define security roles and method permissions in Enterprise beans	"Steps for assembling beans with security roles and method permissions"
Configuring the run-time environment to support the use of security roles or identities	"Steps for configuring the run-time environment for security roles and identities" on page 233

Steps for assembling beans with security roles and method permissions

Application assemblers combine Enterprise beans and other components into J2EE applications that address a business problem or implement a business process. These components may come from diverse sources, so part of the application assembler's job is to integrate these components so that they fit into the customer installation's security domain. Using the WebSphere for z/OS Application Assembly tool, the application assembler defines security roles to control access to bean methods. These roles map to actual user identities or user groups defined to the Security Server (RACF) on the z/OS or OS/390 platform, through the EJBROLE class. Setting up these corresponding RACF class definitions is part of the next procedure, "Steps for configuring the run-time environment for security roles and identities" on page 233.

Before you begin:

- Because security roles in an Enterprise bean's deployment descriptor must be mapped to actual users or user groups on z/OS or OS/390, work with the z/OS or OS/390 security administrator to determine whether currently defined user or group profiles are appropriate, or whether new profiles are required. You will need to use the profile names as names for the security roles you define for the bean.

- Download the latest copy of the Application Assembly tool. For additional details, see "Steps for installing the Application Assembly tool" on page 136.

Perform the following steps to assemble beans with security roles and method permissions, using the WebSphere for z/OS Application Assembly tool:

1. If necessary, define a new application and import the Enterprise bean. If you are modifying a bean that was already imported into an existing application, skip to the next step.

---

2. At the application or bean level, define one security role for each RACF profile that the z/OS or OS/390 security administrator will use for the EJBROLE class. For example, at the application level:
   a. Highlight the application name and click the right mouse button.
   b. Click on `Modify`, then highlight the `Security` tab in the right pane.
   c. Select `Add...` to open a dialog box where you may type in the security role name and an optional description.
   d. Click `OK`, and define as many other security roles as necessary.
   e. Click on the diskette icon to save the definitions.

   Rule: The security role name must exactly match the profile name. Profile names are limited to 245 characters. Blanks are not allowed in the profile name.

---

3. At the bean level, define method permissions for any methods that should have restricted access:
   a. Highlight the bean name and click the right mouse button.
   b. Click on `Modify`, then highlight the `Permissions` tab in the right pane.
   c. Select the security role name, and then scroll through the list of bean methods, clicking on the checkbox to the left of the methods to which you want to restrict access.
   d. Repeat the process with a different security role, as necessary.
   e. Click on the diskette icon to save the definitions.

---

4. If the bean code includes the `isCallerInRole` method, you must link the security role reference in the code to one of the security roles you just defined. Otherwise, skip to the next step.
   a. Highlight the bean name and click the right mouse button.
   b. Click on `Modify`, then highlight the `Security` tab in the right pane.
   c. Select the security role name (which is the reference used in the bean code) and click the `Modify...` button at the bottom of the right pane.

      Tip: If you are working with an Enterprise bean written to the 1.1 specification level, these references will appear automatically under the `Role Name` column.
   d. Click on the down arrow in the `Link` box to display the security roles you just defined, and select the appropriate role to be linked with this role reference.
   e. Repeat the linking process for additional security role references, if any.
   f. Click on the diskette icon to save the definitions.

---

5. (Optional) If you want to define a security identity for WebSphere for z/OS to propagate during the execution of a bean method, modify the RunAs setting for each method. RunAs options are: Caller, server (the default), or security role (any of the roles you have defined).
   a. Highlight the bean name and click the right mouse button.
   b. Click on `Modify`, then highlight the `IBM RunAs` tab in the right pane.
   c. Select the bean method and click the `Modify...` button at the bottom of the right pane.
   d. Click on the down arrow in the `Type` box to select `caller`, `server`, or `role`.
   e. If you selected `role` for the type, click on the down arrow in the `Role` box to display the security roles you just defined, and select the appropriate role to be used for the RunAs identity.
   f. Repeat this modification process for additional methods, if any, for which you want to specify a security identity.
   g. Click on the diskette icon to save the definitions.

---

6. (Optional) If you want to specify that the RunAs identity be used on the operating system thread during method processing:
   a. Highlight the bean name and click the right mouse button.
   b. Click on `Modify`, then highlight the IBM `+ThreadID` tab in the right pane.
   c. Scroll through the list of bean methods, clicking on the checkbox to the left of the methods for which you want to use the RunAs identity on the operating system thread.
   d. Click on the diskette icon to save the definitions.

---

7. Package the bean and other J2EE application components, if any, in an Enterprise Archive (EAR) file. If you need further details, see “Steps for assembling a new J2EE application” on page 137.

When message BBO94010I appears in the status bar, your application is ready to be installed in a WebSphere for z/OS J2EE server, which is one step in the following procedure, “Steps for configuring the run-time environment for security roles and identities.”

Steps for configuring the run-time environment for security roles and identities

If you are installing Enterprise beans for which you want to use security roles, RunAs identities, or a specific RunAs identity on the operating system thread, you need to configure the run-time environment to support these authorization checks. Specifically, you might have to do one or more of the following:

- If the application’s deployment descriptor defines security roles, these roles must be mapped to actual users or user groups on z/OS or OS/390, through the RACF EJBROLE class.
- If the Enterprise beans use RunAs settings to control access to J2EE resources, additional RACF class definitions might be required.
- If bean methods require a specific RunAs setting to be used on the operating system thread, the J2EE server property `Enable Setting OS Thread to RunAs Identity` must be set.

Perform the following steps to configure the run-time environment for using security roles or identities. If you do not have authorization to use RACF commands, you will need the help of the z/OS or OS/390 security administrator to complete some of the following steps:

1. Using RACF commands, complete the following tasks:
   a. Activate the EJBROLE class and allows z/OS or OS/390 systems to share the generic profiles for this class, using the SETROPTS CLASSACT command.
   b. Define the necessary profiles associated with the EJBROLE class, with universal access authority for each profile set to `NONE`, using the RDEFINE command.

      Rule: If the Enterprise bean contains RunAs permissions set to a security role, specify application data through the APPLDATA parameter, which is set to the actual user ID that WebSphere for z/OS is to use for the security role.
   c. Permit read access to each EJBROLE class profile to specific user IDs or groups, using the PERMIT command.
   d. Refresh the EJBROLE class profile information stored in the RACF database, using the SETROPTS REFRESH command.

   Tips:
   - You may use the GEJBROLE class to group security-role profiles.
   - *z/OS Security Server RACF Command Language Reference*, SA22-7687 explains how to submit RACF commands and defines the command syntax and parameters.

2. Make sure that the appropriate user ID (caller, J2EE server region, or security role user ID) has authorization to use system resources that the bean requires, based on the RunAs settings used for the Enterprise bean. To determine what additional authorities might be necessary, review the information in Table 4 on page 33.

3. Using the WebSphere for z/OS Administration application, complete the following tasks to create a new or modify an existing J2EE server configuration:
   a. (Optional) If the application assembler used the `Set OS thread identity to RunAs identity` property, configure the J2EE server to enable setting the operating system thread to the RunAs identity. To do so, highlight the server name and scroll through the properties form until you find the checkbox labelled `Enable Setting OS Thread to RunAs Identity`. Then click on the checkbox.
   b. Install the J2EE applications that use security roles or identities:
      1) Choose `Install J2EE Application...` from the Selected menu bar. The Install J2EE Application dialog box appears.
      2) In the dialog box, enter the fully qualified path name of the EAR file that contains your J2EE application, and the name of the FTP server for the sysplex in which you want to install your application.
      3) Click OK, and the Administration application displays the application content in the EAR file.
      4) Resolve the bean's resource references, if any.

5) After resolving any additional references or resources that this application requires, click OK to start the application installation process.

If you need additional details about installing applications, see “Steps for installing a J2EE application” on page 154.

c. Validate the conversation (that is, the J2EE server configuration) that you have modified or just created. Message BBON0442I appears in the status bar, indicating that the new conversation is valid.
d. Commit the validated conversation. Message BBON0444I appears in the status bar when the new conversation and J2EE server definition are committed.

When you activate this conversation, the J2EE server can successfully perform security checks for bean methods, and use the appropriate security identity during method processing.

Chapter 12. Using the Java Message Service API in J2EE application components

The Java Message Service (JMS) provides a framework for developing and supporting Java software components that communicate by creating, sending, and receiving messages. This method of communication, known as messaging, allows components to interact asynchronously and reliably, without knowing more about their communication partners than message formats and destinations.

WebSphere for z/OS supports the use of the JMS API by the following types of J2EE application components: Servlets, JavaServer Pages (JSPs), and Enterprise JavaBeans.

The following table shows the subtasks and associated information for using the JMS API in the WebSphere for z/OS environment.

Subtask	Associated information (See . . .)
Understanding the concepts related to JMS and the WebSphere for z/OS run-time environment	"Java™ Message Service" on page 45
Completing coding, assembly, and deployment tasks for J2EE applications that use JMS	"Steps for preparing J2EE applications that use the JMS API"
Setting up a J2EE server configuration to support J2EE applications that use JMS	"Steps for configuring JMS resources for the J2EE server" on page 238

Steps for preparing J2EE applications that use the JMS API

Before you can install J2EE applications that use the JMS API into a J2EE server, you need to correctly assemble and deploy the application components. To do so, you need to use the WebSphere for z/OS Application Assembly tool to define WebSphere for z/OS resource types for JMS connection factories and destinations.

Before you begin:

- Note that the following instructions assume that the J2EE applications to be installed have been designed and coded in accordance with the JMS API programming model, as documented in *MQSeries Using Java*, SC34–5456.
- Download the latest copy of the Application Assembly tool. For additional details, see "Steps for installing the Application Assembly tool" on page 136.
- You might want to try the JMS sample that is shipped with WebSphere for z/OS; instructions for setting up and running the sample appear in the accompanying `README` file. The sample and its `README` file are in the HFS directory `/usr/lpp/WebSphere/samples/jms`

Perform the following steps to prepare JMS applications for installation in a J2EE server:

1. Use the Application Assembly tool to make sure that the application deployment descriptor correctly defines a resource reference type for all connection factories and destinations that the application uses.

   Example:

```
<resource-ref>
<description>description</description>
<res-ref-name>jms/TwoPhaseQCF</res-ref-name>
<res-type>javax.jms.QueueConnectionFactory</res-type>
<res-auth>Container</res-auth>
</resource-ref>
```

2. Use the Application Assembly tool to package the J2EE application components in an Enterprise Archive (EAR) file. If you need further details, see "Steps for assembling a new J2EE application" on page 137.

Steps for configuring JMS resources for the J2EE server

If you are installing J2EE applications that use the JMS API, you need to configure their run-time environment (that is, the J2EE server) to include connection factories and destinations. To do so, you need to use the WebSphere for z/OS Administration application to define these connection factories and destinations as J2EE resources for the J2EE server.

If necessary, see *WebSphere Application Server V4.0.1 for z/OS and OS/390: System Management User Interface*, SA22-7838 for more information on how to use the Administration application.

Before you begin: You need to make the following decisions:

- Decide whether to install these J2EE applications in an existing or a new server. The procedure below assumes you are installing the J2EE applications in an existing application server. If you are creating a new server, use the instructions in Chapter 8, "Creating a J2EE server run-time environment," on page 143 along with the information in the following procedure.
- Decide whether you want to enable tracing for MQSeries-related activity. To do so, set the appropriate trace properties in the JVM properties file for the J2EE server. For further information, see:
  - "JVM properties and properties files" on page 339 for instructions about setting up and using a JVM properties file for a J2EE server.
  - *MQSeries Using Java*, SC34–5456, for descriptions of the trace properties you can use for MQSeries.

Perform the following steps to configure JMS connection factories and destinations for an existing J2EE server:

1. Set up MQSeries to work with the WebSphere for z/OS run-time to implement JMS interfaces and to provide administrative and control features.

   Tips:
   - *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834 documents the MQSeries software requirements for use with WebSphere for z/OS.
   - *MQSeries Using Java*, SC34–5456, contains installation instructions for the MQSeries classes for Java and for Java Message Service.

2. Use the Administration application to add MQSeries Java class files to the J2EE server's environment variables:
   - Add the following files to the CLASSPATH environment variable:
     - *mq_install_path*/com.ibm.mqjms.jar
     - *mq_install_path*/com.ibm.mq.jar

– *mq_install_path*

- Add the following to the LIBPATH environment variable: *mq_install_path*

mq_install_path is the HFS where you installed MQSeries; for example: `/usr/lpp/mqm/java/lib`

3. Use the Administration application to define at least one connection factory and destination as J2EE resources. Add these definitions under the J2EE Resources folder, specifying a name and type for each in the properties form.

   Guidelines:

Use this resource type:	If you want:
`MQRRSQueueConnectionFactory` `MQRRSTopicConnectionFactory`	Behavior to be determined by the transactional environment when the JMS client uses the session: • When a global transaction is active, the messaging done in the scope of that transaction becomes part of the logical unit of work. • When a global transaction is not in effect, the session behaves as directed by the specified `transacted` and `acknowledge` modes.
`MQQueueConnectionFactory` `MQTopicConnectionFactory`	Behavior to be determined by only the session's attributes. In this case, the J2EE server ignores the transactional environment in effect when JMS client uses the session.

4. Use the Administration application to define connection factory and destination instances. Add these definitions under the J2EE Resource Instances folder.

   For each resource instance, fill in the appropriate property values. For properties other than those listed in the table below, you may use the default values.

Resource instance:	Property name:	Instructions for supplying a value:
Queue Connection Factory	**Client ID**	Fill in the name of the JMS client (J2EE application) that you will install in this J2EE server.
	Queue Manager Name	Fill in the name of the local MQSeries queue manager to which the JMS client will connect.
Queue	**MQ Queue Name**	Fill in the name of an MQSeries queue that the JMS client will use.
	Queue Manager Name	Fill in the name of the queue manager that will host this queue. Generally, this value is the same as the one you specify for **Queue Manager Name** for the Queue ConnectionFactory.
Topic Connection Factory	**Client ID**	Fill in the name of the JMS client (J2EE application) that you will install in this J2EE server.
	Queue Manager Name	Fill in the name of the local queue manager that is either hosting the message broker or that has channels defined to and from the message broker queue manager.
	Broker Queue Manager	Fill in the name of the queue manager that is hosting the message broker.

Resource instance:	Property name:	Instructions for supplying a value:
Topic	**MQ Topic Name**	Fill in the name of an MQSeries topic that the JMS client will use. **Note:** Unlike queues, topics are created dynamically.

---

5. Use the Administration application to install the J2EE applications that use JMS:
   a. Choose `Install J2EE Application...` from the Selected menu bar. The Install J2EE Application dialog box appears.
   b. In the dialog box, enter the fully qualified path name of the EAR file that contains your J2EE application, and the name of the FTP server for the sysplex in which you want to install your application.
   c. Click OK, and the Administration application displays the application content in the EAR file.
   d. For each application component that uses the JMS API, use the following steps to resolve the connection factory and destination resource references:
      1) Expand the appropriate folder (`EJB Jars` or `Web Apps`) to display the components. When you select a specific component, the Administration application displays information from the deployment descriptor.
      2) Using the Reference and Resource Resolution window, select the J2EE resources you defined earlier in this procedure.
   e. After resolving any additional references or resources that this application requires, click OK to start the application installation process.

   If you need additional details about installing applications, see "Steps for installing a J2EE application" on page 154.

---

6. Validate the conversation that you have modified or just created. Message BBON0442I appears in the status bar, indicating that the new conversation is valid.

---

7. Commit the validated conversation. Message BBON0444I appears in the status bar when the new conversation and J2EE server definition are committed.

When you activate this new conversation, your installed J2EE application may use the connection factories and destinations to send or retrieve messages.

Chapter 13. Using the JavaMail API in J2EE application components

JavaMail provides a framework for developing and supporting Java applications that send, store, and receive mail. Java application components use the JavaMail application programming interface (API) to send and receive mail. The WebSphere for z/OS JavaMail package supports the use of the JavaMail API by the following types of J2EE application components: Servlets, JavaServer Pages (JSPs), and Enterprise JavaBeans.

The following table shows the subtasks and associated information for using JavaMail with WebSphere for z/OS.

Subtask	Associated information (See . . .)
Understanding the concepts related to JavaMail and the WebSphere for z/OS run-time environment	"JavaMail™" on page 49
Completing coding, assembly, and deployment tasks for J2EE applications that use JavaMail	"Steps for preparing J2EE applications that use the JavaMail API"
Setting up a J2EE server configuration to support J2EE applications that use JavaMail	"Steps for configuring mail sessions for the J2EE server" on page 242

Steps for preparing J2EE applications that use the JavaMail API

Before you can install J2EE applications that use the JavaMail application programming interface (API) into a J2EE server, you need to correctly assemble and deploy the application components. To do so, you need to use the WebSphere for z/OS Application Assembly tool to define mail session resources.

Before you begin: Download the latest copy of the Application Assembly tool. For additional details, see "Steps for installing the Application Assembly tool" on page 136..

Perform the following steps to prepare JavaMail applications for installation in a J2EE server:

1. If you have access to the application source code, check to make sure the J2EE application component correctly uses the JavaMail API. For additional information about the JavaMail API, use the reference documentation available through the Sun Microsystems Web site (`http://java.sun.com/`)

   Example: The following code segment shows how the application sends a message and saves it into the mail account's "Sent" folder:

   ```
   javax.mail.Session mail_session = null;
   javax.naming.InitialContext ctx = new javax.naming.InitialContext();
   mail_session = (javax.mail.Session)
      ctx.lookup("java:comp/env/mail/MailSession");

   MimeMessage msg = new MimeMessage(mail_session);
   msg.setRecipients(Message.RecipientType.TO,
        InternetAddress.parse("bob@examplemail.net"));
   msg.setFrom(new InternetAddress("alice@mail.eSeller.com"));
   msg.setSubject("Important message from eSeller.com");
   ```

```
msg.setText(msg_text);
Transport.send(msg);

Store store = mail_session.getStore();
store.connect();
Folder f = store.getFolder("Sent");
if (!f.exists()) f.create(Folder.HOLDS_MESSAGES);
f.appendMessages(new Message[] {msg});
```

---

2. (Optional) If you have access to the application source code, check to see whether the application uses the `setDebug` method to debug JavaMail processing. If not, you may add the method if you want to enable debugging capabilities through the application.

   Example:

   ```
   javax.naming.InitialContext ctx = new javax.naming.InitialContext();
   mail_session = (javax.mail.Session)
      ctx.lookup("java:comp/env/mail/MailSession");
   mail_session.setDebug(true);
   ...
   ```

   Rule: If you add the `setDebug` method to the application, you need to recompile.

   Tip: Even if the application contains the `setDebug` method, you cannot collect diagnostic data through JavaMail's debugging capability unless the J2EE application's run-time environment is set to enable debugging. For instructions on configuring a J2EE server to enable debugging, see "Steps for configuring mail sessions for the J2EE server."

---

3. Use the Application Assembly tool to make sure that the application deployment descriptor correctly defines a resource reference for the mail session, as follows:

   ```
   <resource-ref>
   <description>description</description>
   <res-ref-name>mail/MailSession</res-ref-name>
   <res-type>javax.mail.Session</res-type>
   res-auth<Container></res-auth>
   </resource-ref>
   ```

---

4. Use the Application Assembly tool to package the J2EE application components in an Enterprise Archive (EAR) file. If you need further details, see "Steps for assembling a new J2EE application" on page 137.

Steps for configuring mail sessions for the J2EE server

If you are installing J2EE applications that use JavaMail, you need to configure their run-time environment (that is, the J2EE server) to include mail sessions. To do so, you need to use the WebSphere for z/OS Administration application to define mail sessions as J2EE resources for the J2EE server.

If necessary, see *WebSphere Application Server V4.0.1 for z/OS and OS/390: System Management User Interface*, SA22-7838 for more information on how to use the Administration application.

Before you begin: You need to make the following decisions:

- Decide whether you are going to use the default transport protocol, or a different one. The default transport protocol is SMTP.

- Decide whether to allow J2EE applications to access stored messages. If so, you must complete the optional step (in the procedure below) for setting up IMAP mail accounts. If your Web applications need only to send messages, you do not need to set up an IMAP account.
- Decide whether to install these J2EE applications in an existing or a new server. The procedure below assumes you are installing the J2EE applications in an existing application server. If you are creating a new server, use the instructions in Chapter 8, “Creating a J2EE server run-time environment,” on page 143 along with the information in the following procedure.

Perform the following steps to configure mail sessions for an existing J2EE server:

1. Set up an out-going mail server that uses the appropriate transport protocol. Note the server name (a fully qualified Internet host name) to use later in this procedure.

   If you are using the default SMTP protocol, set up an SMTP server.

   ---

2. (Optional) If the J2EE applications that you are installing need access to stored messages, set up an appropriate server and mail accounts. Each mail account must consist of three elements: A host name, a user ID, and a password. Note these elements to use later in this procedure.

   If you are using the default IMAP protocol, set up an IMAP server and mail accounts.

   ---

3. (Optional) If you want to enable JavaMail’s debugging capabilities, edit an existing or create a new JVM properties file for the J2EE server, and specify the JVM property `mail.debug=true`

   For additional details about this property, see “JVM properties and properties files” on page 339, which also contains information about the placement and content of a JVM properties file.

   ---

4. Use the Administration application to define a mail session as a J2EE resource. Add this definition under the J2EE Resources folder, specifying a name and type in the properties form.

   ---

5. Use the Administration application to define a mail session instance for an existing or new J2EE server. Add this definition under the J2EE Resource Instances folder. The resource instance properties can be divided into two groups: One for mail sending (transport), and the other for mail store access.

   Rule: You must always specify the mail sending (transport) properties `mail transport host` and `mail originator`. The mail store properties are required only if your J2EE applications need to access stored mail.

Property name:	Instructions for supplying a value:
Mail transport protocol	WebSphere for z/OS sets this property to the default value, SMTP. If you decided to use a different protocol and have installed the required service provider, replace the default with the appropriate protocol name.
Mail transport host	This mail transport property defines the name of your mail server. If you are using the default transport protocol, this is also known as the SMTP server. Enter its fully qualified Internet host name.

Property name:	Instructions for supplying a value:
Mail originator	This mail transport property is an Internet e-mail address that by default will appear in the received message as either the "From" field or the "Reply-To". This is the address to which the recipient's reply will be directed. This default can be overridden for individual messages in your application, using the method Message.setFrom().
Mail transport user and **Mail transport password**	These properties are rarely used for the default SMTP protocol. You may leave them blank. However, if you use a transport protocol that requires username/password, use these properties to enter the required values.
Mail store protocol	WebSphere for z/OS sets this property to the default value, IMAP. If you decided to use a different protocol (such as POP3) and have installed the required service provider, replace the default with the appropriate protocol name.
Mail store host	This property value, along with mail store user and password values, represents a valid mail account. **Example:** If the mail account is `john_william@foo.bar.com`, enter `foo.bar.com` for mail store host.
Mail store user and **Mail store password**	These property values, along with mail store host, represent a valid mail account. **Example:** If the mail account is `john_william@foo.bar.com`, enter `john_william` for mail store user, and the password required to access the given mail account.

6. Use the Administration application to install the J2EE applications that use JavaMail:
   a. Choose `Install J2EE Application...` from the Selected menu bar. The Install J2EE Application dialog box appears.
   b. In the dialog box, enter the fully qualified path name of the EAR file that contains your J2EE application, and the name of the FTP server for the sysplex in which you want to install your application.
   c. Click OK, and the Administration application displays the application content in the EAR file.
   d. For each application component that uses the JavaMail API, use the following steps to resolve the mail session resource references:
      1) Expand the appropriate folder (`EJB Jars` or `Web Apps`) to display the components. When you select a specific component, the Administration application displays information from the deployment descriptor.
      2) Using the Reference and Resource Resolution window, select the J2EE resource you defined earlier in this procedure.
   e. After resolving any additional references or resources that this application requires, click OK to start the application installation process.

   If you need additional details about installing applications, see "Steps for installing a J2EE application" on page 154.

7. Validate the conversation that you have modified or just created. Message BBON0442I appears in the status bar, indicating that the new conversation is valid.

8. Commit the validated conversation. Message BBON0444I appears in the status bar when the new conversation and J2EE server definition are committed.

When you activate this conversation, your installed J2EE application may use the mail session to create and send messages, and to access a message store if you have defined mail store properties for this J2EE resource.

Chapter 14. Steps for configuring Web security

Before you begin: You need to understand how Web security is different from EJB security. See “Web Security” on page 36 for a description of these differences. Because of these differences, in addition to using the Application Assembly tool to set up security roles, you must define how you want your Web security implemented by setting properties in the webcontainer.conf file in See Appendix B, “Default webcontainer.conf file,” on page 351 for more information about the webcontainer.conf file properties you will be changing.

Perform the following steps to configure Web security:

1. Update the **WebAuth.UnauthenticatedUserSurrogate** property in the webcontainer.conf file with the SAF UserID under which unauthenticated clients are to execute.

   ____

2. Update the **WebAuth.LoginToken.Expiration** property in the webcontainer.conf file with the number of minutes for which a login is to be valid. When the token expiration period is reached, the user will be forced to authenticate again.

   ____

3. Set the **WebAuth.LoginToken.LimitToSecureConnections** property in the webcontainer.conf file to true if you want a transport constraint to be used for requests that require use of a login token. When this property is set to true, WebSphere for z/OS will only return the cookie over a secure connection. It will also set the ″secure″ bit in the cookie containing the Login Token. Setting the ″secure″ bit in the cookie instructs HTTP Clients, such as a browser, to only send the cookie on requests that are being sent on a secure transport.

   ____

4. Set the **WebAuth.LoginToken.Encrypt** property in the webcontainer.conf file to true if you want the Login Token is to be encrypted.

   ____

5. Set the **WebAuth.SingleSignOn.Enabled** property in the webcontainer.conf file to true if you want a Login Token to be used for multiple applications existing on different WebSphere Application Servers serving as virtual hosts. These virtual hosts must reside within the domain you specified on the **WebAuth.FormBasedLogin.SingleSignOnDomain** property in the webcontainer.conf file.

   ____

6. Update the **WebAuth.SingleSignOn.Domain** property in the webcontainer.conf file with the name of the domain to which a single sign-on is restricted. This domain name will be used when creating HTTP cookies for single sign-on, and determines the scope to which single sign-on applies. For example, a value of austin.ibm.com would allow single sign-on to work between WebSphere Application Server A with virtual host of serverA.austin.ibm.com and WebSphere Application Server B with virtual host of serverB.austin.ibm.com.

   Note: Cross-domain Single Sign On is not supported. A server at austin.lotus.com, and another at austin.ibm.com cannot partipicate in single sign-on

Chapter 15. Creating and deploying Web Services

WebSphere for z/OS v4.0.1 provides initial support for Web Services. This support enables J2EE components to be exposed as Web Services via Simple Object Access Protocol (SOAP) messages being sent to the server as HTTP or HTTPS requests. WebSphere for z/OS uses existing mechanisms for receiving the HTTP and HTTPS requests. Therefore, you can apply policy such as security and workload management to Web Services in the same manner as you do for existing Websphere J2EE components.

WebSphere for z/OS uses the Apache v2.2 SOAP processor in its current implementation. The SOAP processor is responsible for interpretting the content of a SOAP message and for dispatching the request to a particular component within the J2EE server for handling. The SOAP Processor is able to process messages that comply with the SOAP v1.1 protocol. For more information on the SOAP v1.1 specification, go to URL:

```
http://www.w3.org
```

For more information on the Apache SOAP Processor, go to URL:

```
http://www.apache.org
```

HTTP is a stateless protocol. Therefore, SOAP-style Web Services requests from a client to a Service provider over HTTP must be communicated as independent, stateless, request/response interactions. From this, it follows that the type of artifact best suited to provide the implementation of a Web Service is itself a stateless entity. WebSphere for z/OS is a J2EE component server which is able to manage J2EE components in a robust, secure, and scalable manner. In order to provide Web Services that are able to be accessed flexibility and consistently by clients and are able to be managed efficiently by the server, it is suggested that you use Stateless Session EJBs to implement a Web Service. Stateless Session EJBs are able to be managed efficiently by the runtime while providing the semantic that most closely matches the client programming model for Web Service invokers.

WebSphere for z/OS allows many types of artifacts to be exposed as Web Services. Specifically, handlers are provided which allow Enterprise Java Beans - both Session and Entity beans, Java Programs, Bean Scripting Framework (BSF) Scripts, and standard Java Programs to be exposed as services via SOAP over HTTP. With the exception of the Stateless Session Enterprise Bean Handler, these handlers are being deprecated. This means that these handlers will continue to exist in the runtime for compatibility and migration purposes. Support for these handlers will be removed from the product in a future release. It is recommended that any new Web Services be provided as WebSphere Stateless Session EJBs.

| **Note:** WebSphere Application Server for z/OS does not support the dynamic-properties elements in the XML file that defines acustom service. These elements are supported by WebSphere Application Server for distributed platforms. Therefore, if you are porting a Web service from the distributed platform version of the product to the z/OS version, you may need to make the following code change in order to use the externalConfigURL tag to pass initialization parameters to your Web service:

| 1. Create a separate XML file that contains the dynamic-properties elements.

2. Specify the fully qualified name of this file for the externalConfigURL object when you import the Web service application into an Application Server configuration. The resulting XML file for the Web Service will be similar to the following:

```
<applicationserver
         version="2.0"
         xmi=""
         applicationserver="applicationserver.xmi"
         server="server.xmi"
            id="">
      <nodes
               id="">
          <servers
                  type="ApplicationServer">
              <customServices
                         description="Service1-st2tst"
                         displayName="str2tst"
                     classname="str2tst"
                          externalConfigURL="fully_qualified_file_name"
                          enable="true"
                        />
           </servers>
       </nodes>
</applicationserver>
```

3. Change the custom service initialization routine to read from this new file.

This code change causes the **com.ibm.websphere.runtime.CustomService.externalConfigURLKey** property to set the value of the **externalConfigURL** tag in the Custom Service implementation. This property, which is set to one of the following values, is then passed to the CustomService initialize method:

1. null, if nothing is specified in the **externalConfigURL** tag in the custom service XML file. . 2. The **com.ibm.websphere.runtime.CustomService.externalConfigURLKey** property, which is set to the value of the **externalConfigURL** tag in the Custom Service implementation.

In either case, this information should show up during tracing if orb tracing is turned on.

This changed Web service can be used on all WebSphere Application Server platforms.

Deploying an Enterprise application as a SOAP-accessible Web Service

To deploy a resource as a SOAP-accessible Web Service, you must:

1. Create/locate the software resource to be exposed as a service.
2. Use the Application Assembly Tool (AAT) to package the resource code into an Enterprise Archive (EAR) file.
3. Create the SOAP Deployment Descriptor for the service. In order to deploy an artifact as a Web Service, you need to create a Deployment Descriptor which describes the service you are creating. The information contained within the Deployment Descriptor is dependent upon the type of artifact you are exposing, and might include such information as which methods from a Java class you are exposing, connection information for a DB2 database, or a

JavaScript implementation of a service. (See "Specifying the EJB Deployment Descriptor" on page 252 for more information about how to specify Deployment Descriptor information.)

4. Execute the SoapEarEnabler tool to enable your Web Service. This tool adds all of the parts necessary to deploy the service. (See "Using the SoapEarEnabler Tool" on page 252 for more information about this tool.)
5. Install the service-enabled EAR file as you would install any other Enterprise application into the J2EE server.

Note: WebSphere Application Server for z/OS does not support the dynamic-properties elements in the XML file that defines a custom service. These elements are supported by WebSphere Application Server for distributed platforms. Therefore, if you are porting a Web service from the distributed platform version of the product to the z/OS version you may need to make the following code change to use the externalConfigURL tag to pass initialization parameters to your Web service:

1. Create a separate XML file that contains the dynamic-properties elements.
2. Specify the fully qualified name of this file for the externalConfigURL object when you import the Web service application into an Application Server configuration. The resulting XML file for the Web Service will be similar to the following:

```
<applicationserver
          version="2.0"
          xmi=""
          applicationserver="applicationserver.xmi"
          server="server.xmi"
          id="">
        <nodes
       id="">
          <servers
  type="ApplicationServer">
             <customServices
  description="Service1-str2tst"
  displayName="str2tst"
           classname="str2tst"
      externalConfigURL="fully_qualified_file_name"
      enable="true"
  />
    </servers>
         </nodes>
      </applicationserver>
```

3. Change the custom service initialization routine to read from this new file.

This code change causes the property **com.ibm.websphere.runtime.CustomService.externalConfigURLKey** to set the value of the **externalConfigURL** tag in the Custom Service implementation. This property, which is set to one of the following values, is then passed to the CustomService initialize method:

- null, if nothing is specified in the **externalConfigURL** tag in the custom service XML file.
- The property **com.ibm.websphere.runtime.CustomService.externalConfigURLKey**, which is set to the value of the **externalConfigURL** tag in the Custom Service implementation.

In either case, this information should show up during tracing if orb tracing is turned on.

This changed Web service can be used on all WebSphere Application Server platforms.

Specifying the EJB Deployment Descriptor

SOAP Deployment Descriptors provide information to the SOAP Processor about the services that are to be made available to clients. The exact content of a Deployment Descriptor depends on the type of resource that is being exposed. We recommend that you only use Stateless Session beans to implement a service. Then use the EJB Deployment Descriptor to expose this service. This Deployment Descriptor contains the following tags:

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
             id="urn:ejbadder">
   <isd:provider type="com.ibm.soap.providers.WASStatelessEJBProvider"
                scope="Application"
                methods="create add">
   <isd:option key="JNDIName" value="samples/AdderService"/>
   <isd:option key="FullHomeInterfaceName" value="samples/AdderServiceHome"/>
   <isd:option key="ContextProviderURL" value="iiop://localhost:900"/>
   <isd:option key="FullContextFactoryName"
        value="com.ibm.ejs.ns.jndi.CNInitialContextFactory"/>
   </isd:provider>
   <isd:faultListener>org.apache.soap.server.DOMFaultListener</isd:faultListener>
</isd:service>
```

service-urn is the URN that you want to give to a service. All services deployed within a single EAR file must have URNs which are unique within that EAR.

exposed-methods is a space separated list of methods which you wish to expose.

provider-class is **com.ibm.soap.providers.WASStatelessEJBProvider**

jndi-name is the registered JNDI name of the EJB

home-name is the fully qualified class name of the EJB's home.

Using the SoapEarEnabler Tool

The SoapEarEnabler tool is a Java application which is used to enable a set of SOAP services within an Enterprise Application Archive (EAR). Once these services have been enabled, the EAR can be installed into WebSphere Application Server 4.0.1 so that these web services are available. The SoapEarEnabler will guide you through the steps to enable one or more services within an application. Once you have enabled the Web Services, the EAR will need to be installed into the J2EE server.

The SoapEarEnabler tool is contained in the SoapEnabler.jar file, which is located in the <*installroot*/**wc/lib** directory. There are two modes of operation for using this tool: interactive or silent. Before invoking the SoapEarEnabler tool in either mode, make sure that a SOAP Deployment Descriptor has been created for each service to be enabled (see “Specifying the EJB Deployment Descriptor”).

Invoking the SoapEarEnabler tool in interactive mode

To invoke the SoapEarEnabler tool in interactive mode, issue the following command from the command line to change the directory you are pointing to:

```
cd  <install_root>/wc/lib
```

Then invoke the SoapEarEnabler from the command line without any arguments. As illustrated in the following example, the tool will prompt you for all required information:

```
SoapEarEnabler.sh
Please enter the name of your ear file: ../work/stockquote.ear
How many services would you like your application to contain (1...n)? 1
Now prompting for info for service #1:
Please enter the file name of the Deployment Descriptor xml file: ../work/StockQuoteDD.xml
Is this service an EJB (y/n)? n
How many additional classpath entries are required (0...n)? 0
Should this service be secured (y/n)? n
Please enter a context root for your non-secured services (e.g. /soap): /soap
```

Invoking the SoapEarEnabler tool in silent mode

Using silent mode, when you invoke the SoapEarEnabler, you issue the SoapEarEnabler command; followed by the number of arguments you are going to be including; followed by the actual arguments:

```
soapenabler number-of-arguments argument1 argument2 ...
```

The arguments you include must be placed on the command line in the following order:

1. *<ear-file-name>*
2. *<number-of-services>*
3. The following pair of arguments for each service included in the preceding argument:

   <deployment-descriptor-file-name > *<service-is-an-ejb-(y/n)>*

   For any service that is an EJB service, you must immediately follow the *<service-is-an-ejb-(y/n)>* argument with the *<ejb-jar-file-uri-(already-in-ear)>* argument.
4. <number-of-additional-classpath-entries (0, 1, 2...)>
5. For each additional classpath entry specified in the preceding argument, you must include the following argument:

   <classpath-entry-uri-(already-in-ear)>
6. <secure-this-service-(y/n)>

   If you specify ″n″ on this argument, you must also include the following argument:

   <context-root-for-non-secured-services, ex: /soap>

   If you specify ″y″ on this argument, you must also include the following argument:

   <context-root-for-secured-services, ex: /soapsec>

The following is an example of the command line for deploying one EJB as a non-secured service, using silent mode (the line is split here for printing purposes):

```
SoapEarEnabler soap.ear 1d:\junk\xml-soap\java\samples\ejbadder\
    deploymentdescriptor.xml y adderservice-ejb.jar 1 samples.jar n /soap
```

The following is an example of the command line for deploying one EJB as a non-secured service, and one java class as a secured service, using silent mode (the line is split here for printing purposes)

```
SoapEarEnabler soap.ear 2 d:\junk\xml-soap\java\samples\ejbadder\
    deploymentdescriptor.xml y adderservice-ejb.jar 1 samples.jar n d:\
    junk\xml-soap\java\samples\stockquote\
    deploymentdescriptor.xml n 1 samples.jar y /soap /soap-sec
```

The following is an example of the command line for deploying 2 java classes as non-secured services, using silent mode (the line is split here for printing purposes):

```
SoapEarEnabler soap.ear 2 d:\junk\xml-soap\java\samples\stockquote\
    deploymentdescriptor.xml n 1 samples.jar n d:\junk\xml-soap\
     java\samples\addressbook\deploymentdescriptor.xml n 1 samples.jar n
/soap
```

Creating a SOAP client

You can use the client API provided with WebSphere for z/OS to create SOAP clients to access SOAP services, which have been deployed on an OS/390 J2EE server. For a description of all of the SOAP classes and packages that are currently available, see:

`http://xml.apache.org/soap/docs/index.html`

The basic steps for creating a client which interacts with a SOAP service are as follows:

1. Obtain the interface description of the SOAP service, so you know what the signatures of the methods that you wish to invoke are. You can either look at a WSDL file for the service, or directly at its implementation.
2. Create the Call object The SOAP Call object is the main interface to the underlying SOAP RPC code.
3. Set the target URI into the Call object using setTargetObjectURI() Pass in the URN that the service used to identify itself in its Deployment Descriptor.
4. Set the method name that you wish to invoke into the Call object using setMethodName(). This must be one of the methods exposed by the service at the URN given in the previous step.
5. Create any Parameter objects necessary for the RPC call and set them into the Call object using setParams(). Make sure that you have the same number of parameters with the same types as the service is expecting.
6. Execute the Call object's invoke() method and retrieve the Response object. Remember that the RPC call is synchronous, and so may take a while to complete.
7. Check the response for a fault using getFault(), then extract any result or returned parameters.

While most of the providers only return a result, the DB2 stored procedure provider can also return an SQL ResultSet.

Interacting with a SOAP service which is document-oriented requires that you use lower-level Apache SOAP API calls. You must construct an envelope object which contains the contents of the message (including the body and any headers) that you wish to send, and then create a Message object upon which you invoke the send() method to perform the actual transmission.

Note: Because SOAP is a standard, you should be able to use the clients you create with the Apache SOAP API to access services running on any other implementation that adheres to this standard, and vice versa.

Using XML-SOAP for Remote Procedure Calls

The org.apache.soap.rpc package supports performing RPC over SOAP. The XML-SOAP model is as follows:

- The URI of the method call element is used as the object ID on the remote side.
- The client side API has a ″Call″ object (org.apache.soap.rpc.Call) that is filled in with the method name, object ID and parameters.
- The marshalling/unmarshalling of Java datatypes to/from XML is supported by a type mapping registry (see org.apache.soap.encoding.SOAPMappingRegistry), and serialization (org.apache.soap.util.xml.Serializer) and deserialization (org.apache.soap.util.xml.Deserialization) interfaces that marshallers and unmarshallers, respectively, must implement. The built-in encoders/decoders are simply implementations of these interfaces that are preregistered in the SOAPMappingRegistry.

Once a Call object is set up, its invoke (URL, *string*) method can be used to call the method using the URL as the SOAP endpoint to deliver to. The *string* argument is the value of the SOAPAction header. This method returns a Response object (org.apache.soap.rpc.Response) containing the actual response (if any) or an error message if an error occurred during processing.

Securing SOAP Services

For many applications, in order for SOAP to be a viable enterprise level solution, there needs to be a way to secure SOAP transmissions. WebSphere for z/OS includes three options for providing security for SOAP services when using HTTP as the transport:

- HTTP basic authentication
- SSL (HTTPS) connections
- Web Services W3C Digital Signature Support

Since these security options function independently of each other, they can be combined according to the security requirements for each specific service.

Note: The Web Services W3C Digital Signature Support requires JSSE level 1.0.2, and JCE level 1.2.1. This support will be provided SDK PTF UQ61198.

Using HTTP basic authentication

Your IBM HTTP Server documentation describes how to set up HTTP basic authentication. The Apache SOAP implementation includes a method in the org.apache.soap.transport.http.SOAPHTTPConnection class that can be used to set the password for HTTP Basic authentication. See the following Web site for more information about this method:

```
http://xml.apache.org/soap/docs/index.html
```

When setting up this type of security, remember that each service that requires a different access control policy should be deployed as a separate servlet (RPCRouterServlet). For example, if you have an inventory application for which there are the following two SOAP service entries:

```
https://foo.com/inventory/inquiry
https://foo.com/inventory/update
```

Each service should be deployed as a separate servlet (RPCRouterServlet), so that anyone can inquire about the inventory while only the inventory clerks can update the contents. The ′update′ application does not have to know the identity of the requester. It only needs to know that access is granted.

For this example, the IBM HTTP Server should be configured so that the ′inquiry′ servlet is accessible to anyone, while the ′update′ servlet requires authentication based on the HTTP basic authentication (userid/password).

Using SSL Connections

Using SOAP-SEC over SSL for security enables:

- Data in transit to be protected from eavesdropping or forgery by SSL.
- The server identity to be guaranteed by SSL server authentication.
- The client identity to be authenticated through userid and password, which are encrypted by the SSL transport.

To use SOAP-SEC over SSL, you must setup the SSL connections for the IBM HTTP Server. See your IBM HTTP Server Documentation for details of how to setup the SSL server. You will need to use the Key Management Utility to import the following three certificates into your key database:

- The ″soapclient″ certificate
- The ″soapserver″ certificate
- The soapsec.p12 file as your SSL server′s certificate:

These are all X.509 certificates, and are located in the ′key′ subdirectory of the soapsec.war directory that is set up when the soapsamples.ear file is installed on a J2EE server.

Chapter 16. Using Type 4 JDBC Connectors with WebSphere for z/OS

This chapter describes how to define datasources for WebSphere for z/OS that map to Type 4 JDBC connectors provided by software vendors. WebSphere for z/OS provides runtime support to allow Type 4 JDBC connectors to be configured for Enterprise bean or servlet use. Sample code that demonstrates how to use this support to connect to an Oracle9i database through a Type 4 JDBC connector driver is provided beginning with the section, "Sample Datasource XML Template" on page 264.

Type 4 JDBC drivers are direct-to-database pure Java drivers (″thin″ drivers). A Type 4 driver takes JDBC calls and translates them into the network protocol (proprietary protocol) used directly by the DBMS. Thus, client machines or application servers can make direct calls to the DBMS server.

The Type 4 JDBC connector support described in this guide allows an installation to do the following:

1. Define the XML needed to identify a Type 4 JDBC Connector to the WebSphere for z/OS Administration application so the given Type 4 Connector can be configured for application use
2. Define a JDBC Resource Factory that can be invoked to instantiate and return a datasource for the Type 4 JDBC Connector when an application does a Lookup for that particular type of JDBC datasource.

The support does not in any way enable the Type 4 JDBC Connectors to participate in global WebSphere for z/OS transactions like the JDBC connectors that IBM supports can (i.e. DB2 JDBC, IMS JDBC). Therefore, two-phase commit capability is not supported and a rollback of the global transaction does not occur in the case of an error during an operation on a database supported by these vendor-provided connectors. Also, these Type 4 JDBC connectors cannot take advantage of the connection management provided in the connector support on WebSphere Application Server V4.0.1 for z/OS and OS/390, and thus do not gain the resource authentication or connection pooling that comes with this support.

This section does not describe how to install or configure JDBC connectors provided by software vendors. Refer to the product documentation for the specific connector products for information regarding the installation and configuration of those products on the WebSphere Application Server V4.0.1 for z/OS and OS/390 platform.

Be advised that with the J2EE 1.3 support upcoming in the next release of WebSphere for z/OS, applications using the Type 4 JDBC connector support are expected to require a manual redeployment when upgrading from WebSphere for z/OS V4.0.1 to the J2EE 1.3 level of WebSphere for z/OS.

The following table shows the subtasks and associated information for using Type 4 JDBC Connectors with WebSphere for z/OS.

Subtask	Associated information (See . . .)
Understanding the concepts related to using Type 4 JDBC Connectors with WebSphere for z/OS	Refer to the product documentation associated with the specific Type 4 JDBC connector you plan to use.
• Creating the datasource XML template • customizing the datasource XML template • creating the datasource NLS properties file	"Steps for adding an XML Definition for a Type 4 JDBC Connector to WebSphere for z/OS"
• Creating a a resource factory	"Steps for creating a Resource Factory for the Type 4 JDBC Connector" on page 261
• Developing the application to do a ″lookup″ in the JNDI namespace to obtain an instance of the datasource • Specifying a JDBC resource reference name for your Type 4 JDBC connector datasource using the WebSphere for z/OS Application Assembly tool • Defining a Type 4 JDBC datasource with the Administration application, and • Updating the server region CLASSPATH.	"Steps for developing and deploying applications" on page 262
Creating a XML file containing entries for defining a DataDirect Connect JDBC Oracle9i datasource	"Sample Datasource XML Template" on page 264
Creating a datasource NLS properties file for DataDirect Connect JDBC Oracle9i	"Sample NLS Properties File" on page 271
Creating a sample JNDI Lookup Program for DataDirect Connect JDBC Oracle9i	"Sample Type 4 JDBC Connector Application" on page 272
Creating a sample Resource Factory class for DataDirect Connect JDBC Oracle9i	"Sample Resource Factory Class" on page 274

Steps for adding an XML Definition for a Type 4 JDBC Connector to WebSphere for z/OS

Perform the following steps to add an an XML definition for a Type 4 JDBC Connector to WebSphere for z/OS so that the Administration application can then be used to configure the connector:

1. **Create the datasource XML template**

   To correctly define a Type 4 JDBC connector J2EE resource with the WebSphere for z/OS Administration application, you must provide a datasource template so that the Administration application recognizes the J2EE resource type. For Type 4 JDBC connectors, WebSphere for z/OS PTF **UQ90050** provides the file **ModelJdbcDataSource.xml** which you can use as a model to create a J2EE resource template. The ModelJdbcDataSource.xml file can be found in the following HFS directory on the system which is running WebSphere for z/OS:

   ```
   WS390 INSTALL ROOT/samples
   ```

   If you choose the default directory at installation time, this directory is:

   ```
   /usr/lpp/WebSphere/samples
   ```

   To create an xml file for the Type 4 JDBC Connector you want to use, copy the ModelJdbcDataSource.xml file to a new xml file and give the new file a meaningful name that is descriptive of the Type 4 JDBC Connector. It is suggested that you use the following naming format: *xxxxx*JdbcDataSource where *xxxxx* identifies the vendor and particular type of data base the connector accesses. For example, if the Type 4 JDBC connector you are planning to use is DataDirect's Connect JDBC for accessing an Oracle database, name the file DataDirectOracleJdbcDataSource.xml. Once you have created the xml file for the connector you must then customize it as described in 2 below. When doing the customization, it's suggested that when you give the JDBC DataSource connector a name in the XML, you should use the same JDBC DataSource naming convention that you used to name your XML file.

---

2. **Customize the datasource XML template**

   The ModelJdbcDataSource.xml prolog clearly describes the steps you must follow to modify a copy of the model to create a JDBC datasource XML file for the Type 4 JDBC driver you intend to use. In the model, the following XML tags are used to describe a Type 4 JDBC connector J2EE resource to the Administration application (see "Sample Datasource XML Template" on page 264 for an example of a datasource XML template).

   - XML tags for defining the datasource

     The following XML tags are used to define the Type 4 JDBC resource and resource instance:

     <resource>
     An XML tag marking the beginning of a resource definition section.

     <resource_id>
     An XML tag marking the beginning of a resource identifier definition.

     <name>
     This <resource_id> subtag specifies the datasource type. The name you specify on this tag appears in the drop down list of J2EE types on the Administration application.

     <description>
     This <resource_id> subtag provides a description of the resource.

   - XML tags for defining the resource instance

     The following XML tags are included under the `<resource_information>` section in the datasource XML template:

     <resource_instance_name>
     This tag provides a field on the Administration application to specify the resource instance name.

     <descriptive_name>
     Provides a field on the Administration application for specifying the Type 4 JDBC J2EE resource instance name.

     <resource_instance_description>
     Provides a description of the resource instance.

     <resource_instance_parent>
     Identifies the name of the J2EE Resource this resource instance is associated with.

     <system>
     Identifies the name of the z/OS system that the resource instance is associated with.

   - XML tags for defining resource attributes

In the XML at the comment box "Start of DataSource Properties", use the following XML tag and its related sublevel tags to provide Administration application fields for specifying the properties of a Type 4 JDBC datasource:

<resource_attribute id="*id*"
: This tag in conjunction with its sublevel tags identifies a property of the Type 4 JDBC DataSource that the server administrator must specify on the Administration application when defining a J2EE resource. This tag and its related sublevel tags must be repeated for each property required by the Type 4 JDBC Connector being defined to the Administration application.

- Additional <resource_information> tags pertaining to the resource instance:

  <factory_class_name>
  : The fully-qualified Java class name of the Type 4 JDBC resource factory that you will create. See Procedure To Create Resource Factory Class for details about creating a resource factory class for a Type 4 JDBC connector.

  <connector_class_name>
  : The fully-qualified JDBC datasource class name of the Type 4 JDBC driver you intend to use. Refer to the software vendor's documentation for more information.

  <connector_interface_class_name>
  : The name of the Java interface that supports the J2EE resource. Always set this value to **javax.sql.Datasource**.

---

3. **Create the datasource NLS properties file**

   Each J2EE resource supported by the WebSphere for z/OS Administration application requires a datasource NLS properties file. This file is a companion to the datasource XML template and is read during Administration application initialization. It provides NLS translation for the fields displayed on the Administration application for a particular datasource. The file is organized as a set of key value pairs with each key representing a field specified in the datasource XML template and its corresponding translation (the value). This properties file ensures that Administration application fields defined for a particular J2EE resource are translated to the correct language.

   The naming convention for the NLS properties file should be created with the same name as the XML template and the file type should be "properties" (such as; *xxxxx*JdbcDataSource.properties, where *xxxxx* is the vendor name concatenated with the database name).

   The datasource NLS properties file must be saved in ASCII format in the following HFS directory:

   `CBCONFIG/SYSPLEX/resources/templates/`

   In order for system's management to properly read the NLS properties file the permission bits need to be set to `-rw-r--r--` (644).

   An example of a datasource NLS properties file is shown in "Sample NLS Properties File" on page 271.

   Note: If a properties file is not provided, the Administration application when initialized will put out informational message BBON0517I in the message log indicating no properties file was specified and that, by default, English is used. In the case of installations that wish to only use the English translation of the XML, it is therefore an option to skip creating the properties file and let the processing default to English.

Steps for creating a Resource Factory for the Type 4 JDBC Connector

In order to perform a JNDI lookup on a Type 4 JDBC datasource, you must create a resource factory Java class, compile it, and ensure that it is referenced during runtime. In the datasource XML template, you specified a factory class on the <factory_class_name> tag. During deployment, the Administration application registers this factory class with WebSphere for z/OS. Thus, when an application performs a JNDI lookup on a Type 4 JDBC datasource, this resource factory class is invoked during the lookup process to locate the datasource and pass a reference to it back to the application.

The GenericJdbcResourceFactory.java source file that is shipped with PTF UQ90050 is a model resource factory that you must copy and customize for the particular Type 4 JDBC datasource driver that you are using. "Sample Resource Factory Class" on page 274 shows an example of a customized factory class for the DataDirect Connect JDBC Oracle9i datasource.

To create and install a resource factory class for your Type 4 JDBC datasource:

1. Locate the file **GenericJdbcResourceFactory.java** within the following HFS directory on the system which is running WebSphere for z/OS:
   ```
   WS390 INSTALL ROOT/samples
   ```
   If you chose the default directory at installation time this directory is:
   ```
   /usr/lpp/WebSphere/samples
   ```
2. Copy the GenericJdbcResourceFactory.java source file to an HFS working directory so that you can rename it to *aaaaaa***JdbcResourceFactory**, where *aaaaaa* represents a set of characters that uniquely describe the Type 4 JDBC datasource you are using. It is suggested that for *aaaaaa* you use the Type 4 JDBC Connector vendor name concatenated with the type of database (for example, DataDirectOracle).
3. Edit this file and update the following lines of code:
   a. Change the package name to a name of your choice.
   b. Change the class **GenericJdbcResourceFactory** to *aaaaaa***JdbcResourceFactory**.
   c. Rename the constructor public *aaaaaa***JdbcResourceFactory**.
4. Compile the new JDBC factory java file.
   a. Since you must add the ws390srt.jar file to the CLASSPATH of your java compile, the first step is to locate the copy of this file that was installed along with the WebSphere for z/OS installation. This file will be found in the directory WS390 INSTALL ROOT/lib, which, if you use the default installation directory, is /usr/lpp/WebSphere/lib. This jar file contains WebSphere for z/OS classes that are referenced in GenericJdbcResourceFactory.java and which are needed to complete the datasource lookup processing.
   b. Add the local copy of ws390srt.jar to the compile-time CLASSPATH and perform the compile using whatever Java development tool you are working with (such as WebSphere Studio or the javac compiler which ships with the JDK).
5. Create a jar file that contains the new JDBC factory class.
6. Copy the jar file to an HFS directory available to the Server where you intend to use the Type 4 JDBC Connector.
7. Set the permission bits to `-rwxr-xr-x` (755) in order for the application server region to have proper access to the jar files.

8. Use the WS_EXT_DIRS environment variable to define the work directory containing the JDBC resource factory jar file to the appropriate WebSphere for z/OS application server. You can add the WS_EXT_DIRS environment definition to the server using the Administration application when you define your J2EE server and server instance. (See the *WebSphere Application Server V4.0.1 for z/OS and OS/390: Assembling J2EE Applications* manual for information about updating environment variables.)

Steps for developing and deploying applications

Before you begin: You need to make the following decisions:

- Ensure the Type 4 JDBC Connector has been installed according to the vendor's installation instructions
- Ensure your installation has created the required XML that is needed by the Administration application so an instance of the Type 4 JDBC Connector can be configured
- Ensure your installation has created a JDBC Resource Factory for the Type 4 JDBC Connector so an application can do a JNDI lookup for a datasource that is supported by that particular type of connector
- Ensure a current copy of the WebSphere for z/OS Application Assembly tool has been down loaded and is available for use in your installation.
- Decide whether to install these J2EE applications in an existing or a new server. The procedure below assumes you are installing the J2EE applications in an existing application server. If you are creating a new server, use the instructions in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Assembling J2EE Applications*, SA22-7836 along with the information in the following procedure.

1. **Develop your application to do a ″lookup″ in the JNDI namespace to obtain an instance of the datasource**

   The XML support previously discussed enables Type 4 JDBC connector J2EE resources that are configured to WebSphere for z/OS to be registered in the JNDI namespace as datasources. Applications that wish to use Type 4 JDBC connectors thus need to do a ″lookup″ in the JNDI namespace to obtain an instance of the datasource provide by the Type 4 JDBC connector driver. Once the datasource is obtained, the application can then access it to get a connection to the target database manager.

   To locate a Type 4 JDBC connector J2EE resource, an application should code the lookup the datasource by using the following code:

   ```
   ctx.lookup("java:comp/env/jdbc/datasource_name");
   ```

   where `datasource_name` is an arbitrary name you assign to the Type 4 JDBC connector J2EE resource. The `datasource_name` must be specified as resource on the WebSphere for z/OS Application Assembly tool, together with the ″jdbc″ prefix like this; `jdbc/datasource_name`. During server definition and application deployment, the Administration application binds the `datasource_name` to the real J2EE datasource definition that is configured

   The following code segment, shows an example of the technique for looking up an Oracle9i datasource supported by the DataDirect Connect JDBC connector. A complete code sample is included in "Sample Type 4 JDBC Connector Application" on page 272. Sample JNDI Lookup Program: example of

   Example:

```
public void performTest(javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response)
throws java.io.IOException {

OracleDataSource ds =  null;
Context ctx = null;

try
 {
  // Obtain an initialContext object and create a datasource
  ctx = new InitialContext();
  ds = (OracleDataSource) ctx.lookup("java:comp/env/jdbc/DataDirectOracleJDBCDataSource");

  // Proceed to get a connection from the datasource
 }
}
```

2. **Specify a JDBC resource reference name for your Type 4 JDBC connector datasource using the WebSphere for z/OS Application Assembly tool**

   Using the WebSphere for z/OS Application Assembly tool, a resource reference name associated with a Type 4 JDBC connector datasource must be specified on the Resources tab of the servlet or EJB doing the JNDI lookup of datasource. Refer to *WebSphere Application Server V4.0.1 for z/OS and OS/390: Assembling J2EE Applications* for details on how to create a resource reference. Following are some of the Application Assembly tool Reference tab field specifications that need to be completed:

   Reference name
   This is the name used to lookup the J2EE resource in the JNDI namespace. Note that the 'jdbc' prefix is required. In our sample application the reference name to be created in the Application Assembly tool would be `jdbc/DataDirectOracleJDBCDataSource`

   Type The J2EE type used by Type 4 JDBC connector. Set this field to **javax.sql.Datasource**.

   Authentication
   This field is not applicable for Type 4 JDBC connectors; but, you should leave the value set to CONTAINER.

3. **Define a Type 4 JDBC datasource with the Administration application**

   A JDBC datasource must be defined with the Administration application for Type 4 JDBC connectors to be configured.

   The following Administration application steps need to be done in order to define a new datasource:
   - Create a new conversation.
   - Add a J2EE resource. This is where your datasource created in previous steps should be selected.
   - Add a resource instance.

   For more information on these, see *WebSphere Application Server V4.0.1 for z/OS and OS/390: Assembling J2EE Applications*.

   Following are some of the Administration application datasource fields that need to be completed in the Resource instance:

   Resource name
   This is the name you assign to the datasource.

Resource instance name
This name identifies an instance of the datasource.

Resource Attributes
The resource attributes that you specified in the datasource XML template are displayed as fields on the resource instance display. If you specified default values in the XML template, then those values are displayed as well. You can override these values by typing the new value in the appropriate field.

For example, in the XML template for a DataDirect Connect JDBC Oracle9i datasource, the following fields are defined with <resource_attribute> XML tags in the XML template:

```
System ID name of Remote Oracle Database

TCPIP port number of the listener running on the Oracle Database

Name or IP address of the remote Oracle Database

A valid user id for logging onto remote Oracle Database

A valid password for logging onto remote Oracle Database
```

4. **Add the Type 4 JDBC Connector Driver to the server**

   To provide runtime support for a Type 4 JDBC connector drivers, you must specify the WS_EXT_DIRS environment variable for the server and identify the HFS directory containing the jar files provided by the Type 4 JDBC connector driver vendor.

   Refer to the Type 4 JDBC connector product documentation to determine which jar files are required for a specific database driver.

5. **Complete the Administration application conversation by validating, committing, and activating the conversation.**

   See *WebSphere Application Server V4.0.1 for z/OS and OS/390: Assembling J2EE Applications* for more information.

Sample Datasource XML Template

The following XML file contains entries for defining a DataDirect Connect JDBC Oracle9i datasource.

Example:

```
<?xml version='1.0'?>
<!--=== START OF INSTRUCTIONS ==========================================-->
<!--                                                                  -->
<!--                                                                  -->
<!-- File name:  DataDirectOracleJDBCDataSource.xml                   -->
<!--                                                                  -->
<!-- Descriptive name:  Jdbc DataSource xml template for the SM EUI   -->
<!--                                                                  -->
<!--                                                                  -->
<!-- Proprietary statement:                                           -->
<!--                                                                  -->
<!-- Licensed Material - Property of IBM                              -->
<!-- 5655-F31 (C) Copyright IBM Corp. 2002                            -->
<!--                                                                  -->
<!-- All Rights Reserved.                                             -->
```

```
<!--                                                                  -->
<!-- U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or -->
<!-- Disclosure restricted by GSA-ADP schedule contract with IBM Corp.-->
<!--                                                                  -->
<!-- Status = H28W401                                                 -->
<!--                                                                  -->
<!--                                                                  -->
<!-- NOTE: This program may be used, executed, copied, modified and   -->
<!--       distributed without royalty for the purpose of developing, -->
<!--       using, marketing, or distributing.                         -->
<!--                                                                  -->
<!--                                                                  -->
<!-- Jdbc DataSource XML Creation:                                    -->
<!--                                                                  -->
<!--                                                                  -->
<!-- Follow these instructions to create a WAS/zOS Systems Management -->
<!-- EUI xml file for your resource adapter:                          -->
<!--                                                                  -->
<!-- 1.  Copy this model xml and name the new xml file                -->
<!-- "xxxxxxJdbcDataSource.xml" where xxxxxx is a descriptive name    -->
<!-- for your Jdbc data source. For example, DB2JdbcDataSource.xml,   -->
<!-- OracleJdbcDataSource.xml, etc.                                   -->
<!--                                                                  -->
<!-- 2.  In the prologue of the xml, replace the File name            -->
<!-- "xxxxxxJdbcDataSource.xml" with your xml file name.              -->
<!--                                                                  -->
<!-- 3.  Under <resource_id>, change the <name> value from            -->
<!-- "xxxxxxJdbcDataSource" to the name of your xml file without      -->
<!-- the xml suffix.                                                  -->
<!--                                                                  -->
<!-- 4.  Under <resource_instance_name>, change the <descriptive_name>-->
<!-- value by replacing "xxxxxxJdbcDataSource" with the name of your  -->
<!-- xml file without the xml suffix.                                 -->
<!--                                                                  -->
<!-- 5.  Under <resource_instance_description> change the             -->
<!-- <descriptive_name> value by replacing "xxxxxxJdbcDataSource"     -->
<!-- with the name of your xml file without the suffix.               -->
<!--                                                                  -->
<!-- 6.  Under <resource_instance_parent> change the                  -->
<!-- <descriptive_name> value by replacing "xxxxxxJdbcDataSource"     -->
<!-- with the name of your xml file without the suffix.               -->
<!--                                                                  -->
<!-- 7.  Find <factory_class_name> and change the value specified     -->
<!-- from "your-package-name.xxxxxxJdbcResourceFactory" to the fully  -->
<!-- qualified name of your Jdbc Resource Factory. For example,       -->
<!-- "vender.oracle.OracleJdbcResourceFactory".                       -->
<!-- NOTE: Your Jdbc Resource Factory is the factory that JNDI        -->
<!-- Lookup processing will invoke getObjectInstance() on to get      -->
<!-- a fully initialized instance of your Jdbc DataSource.            -->
<!--                                                                  -->
<!-- 8.  Find <connector_class_name> and change the value specified   -->
<!-- from "your-package-name.xxxxxxJdbcDataSource" to the fully       -->
<!-- qualified name of your Jdbc DataSource. For example,             -->
<!-- "vender.oracle.OracleJdbcDataSource".                            -->
<!-- NOTE: In the case of jdbc connectors, your Jdbc DataSource       -->
<!-- class should be the class that implements javax.sql.DataSource.  -->
<!--                                                                  -->
<!-- 9.  For your information, DO NOT CHANGE the value specified for  -->
<!-- <connector_interface_class_name> that was copied from the        -->
<!-- model. This is the generic interface class name for all Jdbc     -->
<!-- DataSources.                                                     -->
<!--                                                                  -->
```

```
<!-- 10. The section of the xml where you should define the resource  -->
<!-- properties required by your Jdbc DataSource in order to define   -->
<!-- a real data source is bracketed by the following comment lines:  -->
<!--                                                                   -->
<!--      ================================================            -->
<!--      Start of DataSource Properties                              -->
<!--      ================================================            -->
<!--                                                                   -->
<!--      ================================================            -->
<!--      End of DataSource Properties                                -->
<!--      ================================================            -->
<!--                                                                   -->
<!-- Locate this section in the xml file and define the properties    -->
<!-- you need. As an example, there is a resource property that       -->
<!-- is already defined by the model. The identifier for this         -->
<!-- property is <resource_attribute id="DatabaseName">. Use this     -->
<!-- property as a guide and then delete it from the set of           -->
<!-- properties.                                                       -->
<!--                                                                   -->
<!-- When defining properties, there are a number of different        -->
<!-- <attribute-type> choices that can be specified: textarea,        -->
<!-- boolean, combobox, listbox, table, ordered_list, and             -->
<!-- name_value_pair. Refer to the <!ELEMENT> definition descriptions -->
<!-- for these items in the frontend of the xml.                      -->
<!--                                                                   -->
<!-- When defining properties, DO NOT specify any input for the       -->
<!-- following attribute related fields: <field_level_help_url>,      -->
<!-- <validate_regex regular_expression=""/>.                         -->
<!--                                                                   -->
<!-- Refer to the documentation of the JDBC driver software vendor    -->
<!-- to determine the specific fields which need to be added in this  -->
<!-- step.                                                             -->
<!--                                                                   -->
<!-- 11. Finally, delete the instructions from the prologue and copy  -->
<!--     this file into the templates directory.                      -->
<!--                                                                   -->
<!-- To use the new xml with the Systems Management EUI, install      -->
<!-- the xml file in the WebSphere z/OS HFS directory:                -->
<!-- <CBCONFIG>/<PLEXNAME>/resources/templates                        -->
<!--                                                                   -->
<!-- For example, using the default value for <CBCONFIG> and a        -->
<!-- <PLEXNAME> of "PLEX1", the directory would be:                   -->
<!-- /WebSphere390/CB390/PLEX1/resources/templates                    -->
<!--                                                                   -->
<!-- NOTE: This file needs to be copied into the directory in         -->
<!--       EBCDIC format.                                              -->
<!--                                                                   -->
<!--=== END OF INSTRUCTIONS ==========================================-->
<!--                                                                   -->
<!--  Change history:                                                 -->
<!--                                                                   -->
<!-- $L0=MDxxxxx H28W401 20020218 PDNW: initial creation              -->
<!--                                                                   -->
<!--========================= PROLOG ================================-->
<!--                                                                   -->
<!--   File name:  DataDirectOracleJDBCDataSource.xml                 -->
<!--                                                                   -->
<!--   Descriptive name:  Jdbc configuration template for the SM EUI. -->
<!--      Provides support for DataDirect's JDBC driver for non-IBM   -->
<!--      databases.                                                   -->
<!--                                                                   -->
<!--                                                                   -->
```

```
<!--===================================================================-->
<!DOCTYPE websphere390resource [
<!ELEMENT websphere390resource (version,
                                resource)>
<!ELEMENT version EMPTY>
<!ATTLIST version initial CDATA #REQUIRED>

<!ELEMENT resource (resource_id,
                    resource_information)>
<!ELEMENT resource_id (name,
                       description?,
                       general_help_url?)>
<!ELEMENT resource_information (resource_instance_name,
                                resource_instance_description,
                                resource_instance_parent,
                                system,
                                resource_attribute*,
                                factory_class_name,
                                connector_class_name,
                                connector_interface_class_name)>
<!ELEMENT resource_instance_name (descriptive_name,
                                  textfield)>
<!ELEMENT resource_instance_description (descriptive_name,
                                         textarea)>
<!ELEMENT resource_instance_parent (descriptive_name,
                                    textfield)>
<!ELEMENT system (descriptive_name,
                  systemchoice)>
<!ELEMENT resource_attribute (descriptive_name,
                              attribute_type,
                              dependency_logic,
                              field_level_help_string,
                              field_level_help_url,
                              factory_field_name,
                              setter_function)>
<!ATTLIST resource_attribute id ID #REQUIRED>

<!ELEMENT attribute_type (textfield | textarea | boolean | combobox | listbox | table | ordered_list)>
<!ELEMENT textfield (selected_value,
                     default_value?,
                     validate_regex)>
<!ATTLIST textfield editable (Y | N) "Y">
<!ELEMENT textarea (selected_value,
                    default_value?,
                    validate_regex)>
<!ELEMENT boolean EMPTY>
<!ATTLIST boolean selected_value (Y | N) #REQUIRED>
<!ELEMENT combobox (value+,
                    selected_value,
                    default_value?)>
<!ELEMENT listbox (value+,
                   selected_value*,
                   default_value*)>
<!ELEMENT table (validate_name_value_pair,
                 name_value_pair*)>
<!ELEMENT ordered_list (validate_name_value_pair,
                        name_value_pair*)>
<!ELEMENT validate_name_value_pair (validate_regex,
                                    validate_regex)>
<!ELEMENT name_value_pair (name,
                           value)>
<!ELEMENT setter_function (function_name,
                           function_parameter_type)>

<!ELEMENT systemchoice (selected_value)>

<!ELEMENT name (#PCDATA)>
<!ELEMENT value (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT general_help_url (#PCDATA)>
<!ELEMENT factory_class_name (#PCDATA)>
<!ELEMENT descriptive_name (#PCDATA)>
<!ELEMENT selected_value (#PCDATA)>
<!ELEMENT default_value (#PCDATA)>
<!ELEMENT dependency_logic EMPTY>
<!ATTLIST dependency_logic dependency_expression CDATA #REQUIRED>
<!ELEMENT validate_regex EMPTY>
<!ATTLIST validate_regex regular_expression CDATA #REQUIRED>
<!ELEMENT field_level_help_string (#PCDATA)>
<!ELEMENT field_level_help_url (#PCDATA)>
<!ELEMENT factory_field_name (#PCDATA)>
<!ELEMENT connector_class_name (#PCDATA)>
<!ELEMENT connector_interface_class_name (#PCDATA)>
<!ELEMENT function_name (#PCDATA)>
<!ELEMENT function_parameter_type (#PCDATA)>
]>
<websphere390resource>
<version initial="1.0"/>
  <resource>
    <resource_id>
      <name>DataDirectOracleJDBCDataSource</name>
      <description>The DataSource to connect to Oracle</description>
      <general_help_url></general_help_url>
    </resource_id>
    <resource_information>

      <resource_instance_name>
        <descriptive_name>DataDirectOracleJDBCDataSource instance name</descriptive_name>
        <textfield>
          <selected_value></selected_value>
          <validate_regex regular_expression=""/>
        </textfield>
      </resource_instance_name>

      <resource_instance_description>
        <descriptive_name>DataDirectOracleJDBCDataSource instance description</descriptive_name>
        <textarea>
          <selected_value></selected_value>
          <validate_regex regular_expression=""/>
        </textarea>
      </resource_instance_description>

      <resource_instance_parent>
        <descriptive_name>DataDirectOracleJDBCDataSource name</descriptive_name>
        <textfield editable="N">
          <selected_value></selected_value>
          <validate_regex regular_expression=""/>
        </textfield>
      </resource_instance_parent>

      <system>
        <descriptive_name>System name</descriptive_name>
        <systemchoice>
```

```
          <selected_value></selected_value>
        </systemchoice>
      </system>

<!--=====================================================-->
<!--          Start of DataSource Properties          -->
<!--=====================================================-->

<!--=====================================================-->
<!-- Property: System ID (SID)                        -->
<!--=====================================================-->
      <resource_attribute id="SID">
        <descriptive_name>System ID name of Remote Oracle Database</descriptive_name>
   <attribute_type>
     <textfield>
            <selected_value></selected_value>
            <default_value>IBM</default_value>
            <validate_regex regular_expression=""/>
          </textfield>
        </attribute_type>
        <dependency_logic dependency_expression=""/>
        <field_level_help_string> SID </field_level_help_string>
        <field_level_help_url></field_level_help_url>
        <factory_field_name>SID</factory_field_name>
        <setter_function>
          <function_name>setSID</function_name>
          <function_parameter_type>java.lang.String</function_parameter_type>
        </setter_function>
 </resource_attribute>

<!--=====================================================-->
<!-- Property: PortNumber                             -->
<!--=====================================================-->
 <resource_attribute id="PortNumber">
        <descriptive_name>TCPIP port number of the listener running on the Oracle Database</descriptive_name>
   <attribute_type>
     <textfield>
            <selected_value></selected_value>
            <default_value>1521</default_value>
            <validate_regex regular_expression=""/>
          </textfield>
        </attribute_type>
        <dependency_logic dependency_expression=""/>
        <field_level_help_string> Port number for connecting to remote Oracle DB</field_level_help_string>
        <field_level_help_url></field_level_help_url>
        <factory_field_name>PortNumber</factory_field_name>
        <setter_function>
          <function_name>setPortNumber</function_name>
          <function_parameter_type>int</function_parameter_type>
        </setter_function>
 </resource_attribute>

<!--=====================================================-->
<!-- Property: ServerName                             -->
<!--=====================================================-->
 <resource_attribute id="ServerName">
        <descriptive_name>Name or IP address of the remote Oracle Database</descriptive_name>
   <attribute_type>
     <textfield>
            <selected_value></selected_value>
            <default_value></default_value>
            <validate_regex regular_expression=""/>
          </textfield>
        </attribute_type>
        <dependency_logic dependency_expression=""/>
        <field_level_help_string> Server containing the remote Oracle DB</field_level_help_string>
        <field_level_help_url></field_level_help_url>
        <factory_field_name>ServerName</factory_field_name>
       <setter_function>
          <function_name>setServerName</function_name>
          <function_parameter_type>java.lang.String</function_parameter_type>
        </setter_function>
 </resource_attribute>

<!--=====================================================-->
<!-- Property: User                                     -->
<!--=====================================================-->
 <resource_attribute id="User">
        <descriptive_name>A valid user id for logging onto remote Oracle Database</descriptive_name>
   <attribute_type>
     <textfield>
           <selected_value></selected_value>
           <default_value></default_value>
           <validate_regex regular_expression=""/>
          </textfield>
        </attribute_type>
        <dependency_logic dependency_expression=""/>
        <field_level_help_string> User id for logging onto the remote Oracle DB</field_level_help_string>
        <field_level_help_url></field_level_help_url>
        <factory_field_name>User</factory_field_name>
  <setter_function>
          <function_name>setUser</function_name>
          <function_parameter_type>java.lang.String</function_parameter_type>
        </setter_function>
  </resource_attribute>

<!--=====================================================-->
<!-- Property: Password                                 -->
<!--=====================================================-->
       <resource_attribute id="Password">
        <descriptive_name>A valid password for logging onto remote Oracle Database</descriptive_name>
   <attribute_type>
     <textfield>
           <selected_value></selected_value>
           <default_value></default_value>
           <validate_regex regular_expression=""/>
          </textfield>
        </attribute_type>
        <dependency_logic dependency_expression=""/>
        <field_level_help_string> Password for logging onto the remote Oracle DB</field_level_help_string>
        <field_level_help_url></field_level_help_url>
        <factory_field_name>Password</factory_field_name>
  <setter_function>
          <function_name>setPassword</function_name>
          <function_parameter_type>java.lang.String</function_parameter_type>
       </setter_function>
      </resource_attribute>

<!--=====================================================-->
<!--          End of DataSource Properties              -->
<!--=====================================================-->

      <factory_class_name>my.datasource.DataDirectOracleJdbcResourceFactory</factory_class_name>
      <connector_class_name>com.ddtek.jdbcx.oracle.OracleDataSource</connector_class_name>
      <connector_interface_class_name>javax.sql.DataSource</connector_interface_class_name>

    </resource_information>
  </resource>
</websphere390resource>
```

Sample NLS Properties File

The following sample NLS properties file shows the default English language translation for displaying the DataDirect Connect JDBC Oracle9i datasource fields on the Administration application.

Example:

```
#=================================================================================================
# File name: DataDirectOracleJDBCDataSource.properties
#
# Descriptive name: Default English language file for WAS/390
# DataDirectOracleJDBCDataSource J2EE Resource template
#
# Proprietary statement:
#
# Licensed Material - Property of IBM
#
# 5655-F31 (C) Copyright IBM Corp. 2000, 2001
# All Rights Reserved.
# U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
# Disclosure restricted by GSA-ADP schedule contract with IBM Corp.
# Status = H28W401
#
#-------------------------------------------------------------------------------------------------
# Translation Notes:
#
# This file is organized as a sequence of key, value pairs.
# The format is key=value. All text in the value part of the
# key should be translated.
#
# For example, in the key
#
# resource_information.system=System name
#
# the string "System name" should be translated. The key
# (resource_information.system) should not be altered since
# it used as the basis for relating the translated text
# with the according field in the resource xml template.
#
# Any kind of URL in this file does not need to be translated.
# The target for these URLs will be translated separately and
# assigned according URLs by information development.
#
#-------------------------------------------------------------------------------------------------
# Change history:
# $L0=MD12689, H28W401 20011121, PDNW: initial version
#=================================================================================================
#-------------------------------------------------------------------------------------------------
# Base resource definition fields
#-------------------------------------------------------------------------------------------------
factory_class_name=Factory class name
connector_class_name=Connector class name
connector_interface_class_name=Connector interface class name
```

```
resource_id.name=DataDirectOracleJDBCDataSource
resource_id.description=The ConnectionFactory that allows for connection to DataDirect Oracle databases.
resource_information.resource_instance_name=DataDirectOracleJDBCDataSource instance name
resource_information.resource_instance_description= DataDirectOracleJDBCDataSource instance description
resource_information.resource_instance_parent=DataDirectOracleJDBCDataSource name
resource_information.system=System name
#-------------------------------------------------------------------------------------------------
# NLS Definition for resource attribute ==>SID<==
#-------------------------------------------------------------------------------------------------
resource_information.resource_attribute.SID.descriptive_name=System ID Name of Remote Oracle Database
resource_information.resource_attribute.SID.field_level_help_string= \
The SID property refers to the instance of the Oracle database software running on the server. \
The default value is ORCL.
#-------------------------------------------------------------------------------------------------
# NLS Definition for resource attribute ==>PortNumber<==
#-------------------------------------------------------------------------------------------------
resource_information.resource_attribute.PortNumber.descriptive_name= \
TCPIP port number of the listener running on the Oracle Database
resource_information.resource_attribute.PortNumber.field_level_help_string= \
The PortNumber property identifies the port used to connect to the remote Oracle database.
#-------------------------------------------------------------------------------------------------
# NLS Definition for resource attribute ==>ServerName<==
#-------------------------------------------------------------------------------------------------
resource_information.resource_attribute.ServerName.descriptive_name= \
Name or IP address of the remote Oracle Database
resource_information.resource_attribute.ServerName.field_level_help_string= \
The ServerName property identifies the server containing the remote Oracle database.
#-------------------------------------------------------------------------------------------------
# NLS Definition for resource attribute ==>User<==
#-------------------------------------------------------------------------------------------------
resource_information.resource_attribute.User.descriptive_name= \
Valid user id for logging onto the remote Oracle Database
resource_information.resource_attribute.User.field_level_help_string= \
The User property identifies the user id for logging onto the remote Oracle database.
#-------------------------------------------------------------------------------------------------
# NLS Definition for resource attribute ==>Password<==
#-------------------------------------------------------------------------------------------------
resource_information.resource_attribute.Password.descriptive_name= \
Valid password for logging onto the remote Oracle Database
resource_information.resource_attribute.Password.field_level_help_string= \
The Password property identifies the password for logging onto the remote Oracle database.
#=================================================================================================
# End of File
#=================================================================================================
```

Sample Type 4 JDBC Connector Application

The following example shows an application that was written to Lookup a DataDirect Connect JDBC Oracle9i datasource, obtain a connection to the data base, and then retrieve data from the data base and display it.

Example:

```
package com.ibm.oracle;
import javax.servlet.SingleThreadModel;
import javax.servlet.http.HttpServlet;
import java.sql.*;
import javax.sql.*;
import java.io.*;
import javax.naming.*; // Required for Initial context
import javax.transaction.xa.*;
import com.ddtek.jdbcx.oracle.OracleDataSource;

// This servlet does a JNDI lookup of a DataDirect Oracle JDBC datasource
// and then gets a connection from that datasource
public class TestLookupOra
    extends HttpServlet
    implements SingleThreadModel
{

    public void doPost(
        javax.servlet.http.HttpServletRequest request,
        javax.servlet.http.HttpServletResponse response)
    throws javax.servlet.ServletException, java.io.IOException
    {
        performTest(request, response);
    }

    public void doGet(
        javax.servlet.http.HttpServletRequest request,
        javax.servlet.http.HttpServletResponse response)
    throws javax.servlet.ServletException, java.io.IOException
    {
        performTest(request, response);
    }

    public void performTest(
        javax.servlet.http.HttpServletRequest request,
        javax.servlet.http.HttpServletResponse response)
    throws java.io.IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        OracleDataSource ds = null;
        Context ctx = null;
        java.sql.Connection myCon = null;

        try
        {
            // Obtain an initialContext object and create a datasource
            ctx = new InitialContext();
            ds = (OracleDataSource) ctx.lookup("java:comp/env/jdbc/DataDirectOracleJDBCDataSource");

            // The datasource properties: SID, PortNumber,
            // ServerName, User and Password are set by
            // the application deployer using the SMEUI so
            // they do not need to be set explicitly here by the application
            // developer by driving setters on the datasource object.
            // Now that the JNDI lookup is complete, these properties have
            // already been set on the datasource.

            // Get connection from datasource.
            // We could also have used getConnection(userid,password).
            myCon = ds.getConnection();

            // Use connection.  Use the myCon JDBC connection to
            // perform reads and/or writes to the database. We do not
            // show examples of reads or writes in this sample application
            // but leave this up to the reader's imagination.

            // Close connection when we are done with it.
            myCon.close();

            // Display connection success msgs
            System.out.println("Success! -- Connection established by JNDI lookup.");
            out.println("<HTML>");
            out.println("<HEAD><TITLE>Oracle9i JNDI Lookup Connection Test</TITLE></HEAD>");
            out.println("<BODY>");
            out.println("<P>");
            out.println("Success! Connection established by JNDI lookup.");
            out.println("</BODY></HTML>");
        } catch (Exception ex)
        {
            ex.printStackTrace();
            out.println("<HTML>");
            out.println("<HEAD><TITLE>Oracle9i JNDI Lookup Connection Test Failed</TITLE></HEAD>");
            out.println("<BODY>");
            out.println("<P>");
            out.println("Connection failed! -- see stack trace.");
            out.println("</BODY></HTML>");
        }
    }
}
```

Sample Resource Factory Class

The following example contains code to define the resource factory class for a DataDirect Connect JDBC Oracle9i datasource.

Example:

```
package my.datasource;

//------------------------------------------------------------------------
//
//  Module name:  DataDirectOracleJdbcResourceFactory
//
//  Descriptive Name: WS/390 DatatDirect Oracle Jdbc Resource Factory
//
//  Proprietary statement:
//
// Licensed Material - Property of IBM
// 5655-F31 (C) Copyright IBM Corp. 2002
//
// All Rights Reserved.
//
// U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
// Disclosure restricted by GSA-ADP schedule contract with IBM Corp.
//
// Status = H28W401
//
// NOTE: This program may be used, executed, copied, modified and
//       distributed without royalty for the purpose of developing,
//       using, marketing, or distributing.
//
//  Change activity:
//
// Flag  Reason   Release   Date   Pgmr          Description
// ---- --------- ------- -------- ----- ------------------------------
// $L0= Jdbc      H28W401 20020318 PDNW: initial creation
// $L1= Oracle    H28W401 20020423 PDMU: Customized for DataDirect
//                                       Oracle driver
//------------------------------------------------------------------------

// These import statements are not necessary, since for instructional
// purposes the fully-qualified class names were used in the sample code
// below.  We leave them in as a reminder of which packages are used here.

import java.io.BufferedWriter;
import java.io.PrintWriter;
import java.util.Enumeration;
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.Name;
import javax.naming.Reference;
import javax.naming.spi.ObjectFactory;
import javax.sql.DataSource;

// Referenced classes of package com.ibm.ws390.container.resref:
//
//            BaseResourceFactory, ResourceRefAddr, ResourceInfo
//
// These classes can be found in ws390srt.jar which ships along
// with WebSphere for z/OS.  See instructions for details.

import com.ibm.ws390.container.resref.BaseResourceFactory;
import com.ibm.ws390.container.resref.ResourceRefAddr;
import com.ibm.ws390.container.resref.ResourceInfo;


// Class Defintion

public class DataDirectOracleJdbcResourceFactory
  extends com.ibm.ws390.container.resref.BaseResourceFactory
    implements javax.naming.spi.ObjectFactory {

 public DataDirectOracleJdbcResourceFactory() {
 }

 public java.lang.Object getObjectInstance(java.lang.Object obj,
                                              javax.naming.Name name,
                                              javax.naming.Context context,
                                              java.util.Hashtable hashtable)
throws java.lang.Exception
 {

  com.ibm.ws390.container.resref.ResourceRefAddr resourcerefaddr = null;
  com.ibm.ws390.container.resref.ResourceInfo resourceinfo = null;
  java.lang.String j2eeResourceName = null;
  java.lang.String logwriterParm = null;

  javax.naming.Reference reference = (javax.naming.Reference) obj;
  java.util.Enumeration enumeration = reference.getAll();

  // Create the Jdbc DataSource

  System.out.println("Oracle resource factory: Getting datasource class");
  java.lang.Class datasourceClassName = java.lang.Class.forName(reference.getClassName());

  System.out.println("Oracle resource factory: Create JDBC datasource");
  java.lang.Object jdbcDataSource = datasourceClassName.newInstance();

  //  Set the properties from the Systems Management EUI on the resource

  for (; enumeration.hasMoreElements();
      com.ibm.ws390.container.resref.BaseResourceFactory.driveSetter(resourcerefaddr, jdbcDataSource))
      {
      resourcerefaddr = (com.ibm.ws390.container.resref.ResourceRefAddr) enumeration.nextElement();
  }

  //  Return the datasource now that the properties have been set
  return jdbcDataSource;
 }

}
```

Part 4. Working with J2EE applications in the run-time environment

Chapter 17. Installing applications in a WebSphere for z/OS server

In addition to step-by-step installation through the WebSphere for z/OS Administration application, you may use the following alternative methods of installing applications in a WebSphere for z/OS:

For information about:	See . . .
Using the export/import function of the Administration application	"Steps for using the export/import process through the Administration application"
Using the System Management Scripting APIs	"Installing applications using scripts" on page 280

Steps for using the export/import process through the Administration application

After you have finished testing your J2EE applications, you can use the WebSphere for z/OS Administration application to export the J2EE server configuration you have been using on your test system, and import that model configuration on a production system. Through this export/import process, you create an HFS file that contains the server definition, which you transfer to a production system. This process can be quicker and less error-prone than defining a server configuration from scratch.

Perform the following steps to use the export/import process:

1. In the Administration application, export the server model of the J2EE server in which your application is deployed:
   a. Select the server in the active image.
   b. Select the **export server...** action of the Selected menu bar choice. The **Export server** dialog box appears.
   c. Enter the fully qualified name of an HFS file to contain the output of the export process.
   d. Click **OK**.

   Result: The action Export server... creates HFS files for the server on the host. These files contain definitions of the server and its subtree with almost all its properties, even referenced but not defined J2EE resources.

2. Copy or move the output HFS files to the z/OS or OS/390 production system on which you want the server to run. See *z/OS UNIX System Services User's Guide*, SA22-7801 for methods of and instructions for moving or copying files.

   Warning: Do not edit the output HFS file.

3. In the Administration application, import the J2EE server model by completing the following steps:
   a. Add a conversation, if necessary.
   b. Select the **Servers** folder.

© Copyright IBM Corp. 2000, 2003

c. Select the **import server...** action of the Selected menu bar choice. The **Import server** dialog box appears.
d. For **Server name**, enter a name that is unique to this WebSphere for z/OS configuration.
e. For **Input file**, enter the fully qualified name of the HFS files that you moved or copied to the production system.
f. Click **OK**.
g. Modify the properties of the server, including **Control region proc name** and **Debugger allowed**.
h. Add server instances for the production system, as appropriate.
i. Add J2EE resource instances for the production system, as appropriate.

---

4. Also in the Administration application:
   a. Validate the imported model by selecting the conversation, then selecting **Validate**. When message BBON0442I appears in the status bar, the new conversation is valid.
   b. Commit the conversation by selecting the conversation, then selecting **Commit**. Answer Yes to the question: ″Do you still want to commit?″ When message BBON0444I appears in the status bar, the new conversation was committed.
   c. Complete z/OS or OS/390 tasks, as appropriate.
   d. Activate the conversation by selecting the conversation, then selecting **Activate**. Answer Yes to the question: ″Do you want to activate conversation... ?″ At the bottom of the dialog, a message indicates when the server definition has been activated.

Installing applications using scripts

To install applications in a J2EE server without using the WebSphere for z/OS Administration application, you may use the System Management Scripting APIs, which provide similar capabilities as the Administration application. Using the scripts might provide a quicker, less error-prone method of installing applications into a production server, for example. For more information about using the System Management Scripting APIs, see *WebSphere Application Server V4.0.1 for z/OS and OS/390: System Management Scripting API*, SA22-7839.

Chapter 18. Collecting data about J2EE application activity

WebSphere for z/OS offers several different methods of collecting information about applications running in a J2EE server:

For information about:	See . . .
Using SMF records to collect accounting information	"Collecting J2EE application information through SMF records"
Using the IBM Distributed Debugger to collect diagnostic data for distributed applications	"Debugging and tracing distributed applications"
Using JRas support to enable applications to issue messages and trace entries	"Logging messages and trace data for Java applications" on page 284

Collecting J2EE application information through SMF records

If you want to collect and record statistics related to your server applications, you may define a J2EE server to use the z/OS or OS/390 systems management facility (SMF). Through SMF activity and interval records, the J2EE server records application details that you may use for application profiling. To enable SMF recording, you must define the J2EE server to create SMF records, and perform other administration tasks; for further details, start with the SMF topic in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Operations and Administration*, SA22-7835.

Debugging and tracing distributed applications

The IBM Distributed Debugger and Object Level Trace tools enable you to monitor and debug distributed applications, including the components that run in an WebSphere for z/OS server. The IBM Distributed Debugger and Object Level Trace provide debugging and tracing capabilities for Java or C++ application components and their Java or C++ clients, which may reside on platforms other than z/OS or OS/390.

The following table shows the subtasks and associated information for using the IBM Distributed Debugger and Object Level Trace tools, which are hereafter called Debugger and OLT, respectively.

Subtask	Associated information (See . . .)
Learning concepts related to the Debugger and OLT	The **InfoCenter** for WebSphere V3.5 or V4.0 for distributed platforms. The InfoCenter is available at: `http://www.ibm.com/software/webservers/appserv/`
Installing the Debugger and OLT on your workstation	The **InfoCenter** as listed above

© Copyright IBM Corp. 2000, 2003

Subtask	Associated information (See . . .)
Setting up the workstation and z/OS or OS/390 environments and applications for using the tools	• "Steps for starting the Debugger and OLT on your workstation" • "Steps for preparing the Debugger and OLT for Windows Java clients" on page 283 • "Step for preparing z/OS or OS/390 Java clients" on page 283 • "Steps for preparing J2EE application components in a WebSphere for z/OS J2EE server" on page 283
Using the Debugger and OLT interfaces and output	The **InfoCenter** as listed above

Steps for starting the Debugger and OLT on your workstation

Perform the following steps to start the Debugger and OLT on a Windows NT or 2000 workstation:

1. Start the OLT Viewer from either the Windows Taskbar Start Menu, or type `olt` from an MS-DOS command prompt.

2. Start the Browser Preferences window by selecting **File → Preferences...**

3. From the Browser Preferences window, click on **OLT**. Write down the OLT Server TCP/IP port value (the default is 2102), which is the value you will later specify for the client environment variable.

4. From the Client Controller, in the Execution Mode list box, set the application execution mode to one of the following:
   - `Trace only`
   - `Debug only`
   - `Trace and debug`
   - `No trace and debug`

   Make sure you hit the Apply button to save any changes.

   Note: If the execution mode is set to `Debug only` or `Trace and debug`, the debugger host name and debugger TCP/IP port field is enabled. You can change the debugger host name and port to values that reflect the location of the Debugger. This host name should be the same as the value you specify for the OLT Server host name if OLT and the debugger interface run on the same machine. Otherwise, these host name values will be different. If your installation does not have DNS configured for the WebSphere for z/OS environment, make sure you use an IP address as the Remote Debugger host name.

   The default setting for the Debugger host name is the local host name, and the default for the Debugger TCP/IP port is 8001.

Before continuing to the next procedure, make sure that you remember the following:
- The monitoring mode you selected for debugging.

- The IP addresses and port numbers for the machines on which the OLT server and OLT client controller are running.

Steps for preparing the Debugger and OLT for Windows Java clients

To prepare the Debugger and OLT for Java clients that run on Windows and use Java application components that are installed in a J2EE server, perform the following steps:

1. Create the client startup command based on the following default startup command, replacing the italicized variables in the properties highlighted in bold type, as instructed in the following steps:

   ```
   java -Xdebug -Xnoagent -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=address
   -Djava.compiler=NONE -Dcom.ibm.debug.jdwpport=address
   -Xbootclasspath/a:%JAVA_HOME%\lib\tools.jar -classpath %SOMCBASE%\lib\somojor.zip;
   %SOMCBASE%\samples\InstallVerification\ProgrammingModel\BusinessObjects\Policy\
       Working\NT\TRACE_DEBUG\JCB\jcbPolicyC.jar;
   %SOMCBASE%\lib\dertrjrt.jar;%CLASSPATH%
   -Dcom.ibm.CORBA.EnableApplicationOLT=true
   -Dcom.ibm.CORBA.OLTApplicationHost=hostname
   -Dcom.ibm.CORBA.OLTApplicationPort=portnumber -DOLTClient=true
   -Dcom.ibm.CORBA.BootstrapHost=bootstrap_hostname PolicyApp [options]
   ```

2. Change `address` and `jdwpport` to the same value that you will use for the JVM_DEBUG_PORT variable for the J2EE server in which the Java application components are installed.

3. Change the `OLTApplicationHost` to the hostname where the OLT runs.

4. Change the `OLTApplicationPort` to the OLT server's TCP/IP port.

5. Change the `BootstrapHost` to the host name where the WebSphere for z/OS J2EE server runs.

Step for preparing z/OS or OS/390 Java clients

To prepare Java clients that run on z/OS or OS/390 and use Java application components that are installed in a J2EE server, perform the following step:

- Add the following environment variables to the Java client's shell script:

  ```
  export HOME=/tmp
  export IVB_DEBUG_ENABLED=1  # enable the OLT tools
  export IVB_TRACE_PORT=2102  # OLT Server TCP/IP port
  export IVB_TRACE_HOST=address # the IP address where the OLT is running,
                                # or if WebSphere for z/OS has DNS set up,
                                # you may use the host name for
                                # IVB_TRACE_HOST
  ```

Steps for preparing J2EE application components in a WebSphere for z/OS J2EE server

Before you begin: Find out whether the IBM Distributed Debugger and Object Level Trace are installed and available at your installation.

Follow these instructions, from the workstation where you installed the Debugger and OLT, to enable the Java application components for OLT:

1. Start the WebSphere for z/OS Administration application and create a new conversation for a new J2EE server.

2. Expand the J2EE servers folder and highlight the J2EE server. Then, in the server properties form:
   - Check the `Debugger allowed` check box.
   - Set the `Object Level Trace Hostname` to the name of the machine where OLT is running, and set the `Object Level Trace Port` to the value you set in "Steps for starting the Debugger and OLT on your workstation" on page 282. (The default port is `2102`).
   - Add the following environment variable only for Java business objects:
     ```
     JVM_DEBUG_PORT=xxxx
     ```
     where *xxxx* is the port number that the Debugger will use to connect to the running JVM.

Logging messages and trace data for Java applications

The WebSphere for z/OS run-time supports the Ras Toolkit for Java, which enables you to issue messages from and collect trace data for your Java server applications that run in WebSphere for z/OS J2EE or MOFW servers. Through WebSphere for z/OS extensions to the toolkit, known as JRas support, your Java application's messages can appear on the z/OS or OS/390 master console or in the error log stream, depending on the message type. All messages are logged in the component trace (CTRACE) data set for WebSphere for z/OS. Also, your application's trace entries can appear in the same CTRACE data set.

You might want to issue messages to the master console to report serious error conditions for mission-critical applications. Through the master console, an operator can receive and, if necessary, take action in response to a message that indicates the status of your application. In addition, by directing messages to the master console, you can trigger automation packages to take action for specific conditions or events related to your application's processing.

With JRas support, you may direct error messages to the error log stream. Any messages that your application issues also appear in the CTRACE data set for WebSphere for z/OS. Logging the messages in these system resources can help you more easily diagnose errors related to your application's processing.

Similarly, issuing requests to log trace data in the CTRACE data set is another method of recording error conditions, or collecting application data for diagnostic purposes. You can select the amount and types of trace data to be collected, so you have the ability to run your application with minimal tracing, when performance is a priority, or to run your application with detailed tracing, when you need to recreate a problem and collect additional diagnostic information.

Recommendation: The error log stream, the CTRACE data set for WebSphere for z/OS, and the master console are primarily intended for recording diagnostic data for or monitoring system components and critical applications. Depending on your installation's configuration, directing application messages and data to these resources might have an adverse affect on system performance. For example, if you send application data to the CTRACE data set, trace entries in that data set might wrap more quickly, which means you might lose some critical diagnostic data because the system writes new entries over existing ones when wrapping occurs. Use this logging support judiciously.

Notes:

1. You can use the WebSphere for z/OS support for logging messages and trace data only for Java applications (not for Java applets).
2. The WebSphere for z/OS support for the Ras Toolkit is not the same as the JRas support supplied in Enterprise Edition V3.02. The new JRas support:
   - Always logs messages that your application issues. This change means that, once you code an application to issue messages and run that application, its messages will always be collected and logged. With Enterprise Edition V3.02, you had the ability turn off message collection.
   - Requires a different mechanism for enabling the collection of trace data. With Enterprise Edition V3.02, environment variables for the MOFW application server controlled the collection of trace data; with WebSphere for z/OS V4.0, a customer-supplied trace settings file enables or disables the collection of trace data.
   - Uses different classes for obtaining message or trace loggers, but the same methods: the createRASTraceLogger and createRASMessageLogger methods. The WebSphere for z/OS V4.0 methods, however, have slightly different signatures than those for Enterprise Edition V3.02.

     Although the Enterprise Edition V3.02 createRASTraceLogger and createRASMessageLogger methods are deprecated, you do not have to change any of the programs you coded to use them, unless those programs must run on another platform as well as on z/OS or OS/390. With WebSphere for z/OS V4.0, calls to createRASTraceLogger or createRASMessageLogger are delegated to the same methods in the new WebSphere for z/OS V4.0 class. To run your application on additional platforms, such as Windows NT, you must recode your program to use the new methods.

     For descriptions of the methods you can issue from your server application
| to issue messages or log trace entries, refer to the InfoCenter at
| `http://www.ibm.com/software/webservers/appserv/library.html`. The InfoCenter describes the JRas Facility methods in the `com.ibm.ras` package, as it applies to all supported platforms, including z/OS or OS/390.

The following table shows the subtasks and associated procedures for logging messages and trace data for your Java application:

Subtask	Associated procedure (See . . .)
Determining which types of messages and trace data to issue or collect	• “Background on issuing application messages to the z/OS or OS/390 master console” on page 286 • “Background on issuing trace requests for your application” on page 287
Preparing your Java server application to issue messages and trace requests	“Steps for coding your Java application to issue messages and trace requests” on page 289
Preparing the z/OS or OS/390 run-time environment for logging messages and collecting trace data	“Steps for preparing the z/OS or OS/390 environment for logging Java application messages and trace requests” on page 294

Subtask	Associated procedure (See . . .)
Viewing messages or trace data collected for your Java server application	• "Background on viewing messages and trace data" on page 296 • "Steps for using IPCS in batch mode to format application trace data" on page 297

Background on issuing application messages to the z/OS or OS/390 master console

With the WebSphere for z/OS run-time support for the Ras Toolkit (JRas support), you can issue messages from your Java application to the master console. You might want to issue messages to the master console to report serious error conditions for mission-critical applications, or to trigger automation packages. The messages your application issues also appear in the component trace (CTRACE) data set that WebSphere for z/OS uses, and in its error log stream if the messages are classified as error messages. Logging the messages is another method of recording error conditions, or collecting application data for diagnostic purposes.

WebSphere for z/OS provides code that creates and manages a message logger, which processes your application's messages. The message logger runs in the Java virtual machine (JVM) for the WebSphere for z/OS J2EE or MOFW server in which your Java application will run. To use a message logger, all you need to do in your Java application is:

1. Define the message logger,
2. Drive the method to instruct WebSphere for z/OS to create the message logger, and
3. Code messages at appropriate points in your application. To direct specific messages to the master console, your code must include the appropriate classification for each message.

Specific instructions for updating your application to use JRas support appear in "Steps for coding your Java application to issue messages and trace requests" on page 289. Before you can use those instructions to properly code messages, however, you need to understand the concepts in the following topics:
- "Defining messages through inline method calls or a message properties file"
- "Understanding how the message type affects message destinations" on page 287

Defining messages through inline method calls or a message properties file

If you want to issue messages from your Java application, you may either define the messages inline, or use a separate file to contain the messages. Generally speaking, defining messages inline is faster and requires fewer steps to complete; using a separate message properties file is a better approach for both usability and for text translation, if you plan to provide message text in a variety of languages. Regardless of whether you use the file or inline approach for defining messages, you must code methods in your Java application to issue messages at appropriate points in its processing. At those points, you use methods defined in the `RASIMessageLogger` interface to issue messages.

If you define messages inline, use `textMessage` methods to issue messages from your application. The string that you specify on the method call is what the message logger sends to the master console, error log stream, or CTRACE data set.

If you plan to use a message properties file, you need to:

1. Create the message properties file.
2. Define all messages using a key/text pair.

   The key enables the message logger to locate the appropriate message in the message file; the text is what the message logger sends to the master console, error log stream, or CTRACE data set.
3. Use the appropriate methods to tell the message logger where to find message text for your application's messages.

   You can identify the message file to the message logger through two mechanisms:
   - The `setMessageFile` method, which registers one message properties file to serve as the default file for retrieving message text.
   - The `message` or `msg` methods, on which you may specify the name of the message properties file.

See "Steps for coding your Java application to issue messages and trace requests" on page 289 for specific instructions for creating a message file, rules for defining the messages in it, and examples.

Understanding how the message type affects message destinations

When you code the method to issue a message, you assign a message type to characterize the message as an error, warning, or informational message. The `RASIMessageEvent` interface defines the message types. These types define the destination of each message:

- Only informational messages (`TYPE_INFORMATION` or `TYPE_INFO`) are sent to the master console.
- Only error messages (`TYPE_ERROR` or `TYPE_ERR`) are sent to the error log stream.
- All three types of messages are sent to the CTRACE data set.

Note that messages are always logged; once you code an application to issue messages, and run that application on z/OS or OS/390, its messages will always be collected and logged.

Background on issuing trace requests for your application

The purpose of collecting trace data is to provide sufficient information to diagnose a problem with your application. With the WebSphere for z/OS run-time support for the Ras Toolkit (JRas support), you can issue trace requests from your Java application, and have the resulting trace data recorded in the component trace (CTRACE) data set that WebSphere for z/OS uses. Your application's trace data appears in the CTRACE data set for the WebSphere for z/OS J2EE or MOFW server in which your application runs.

WebSphere for z/OS provides code that creates and manages a trace logger, which processes your application's trace requests. The trace logger runs in the Java virtual machine (JVM) for the WebSphere for z/OS J2EE or MOFW server in which your Java application will run. To use a trace logger, all you need to do in your Java application is:

1. Define the trace logger,
2. Drive the method to instruct WebSphere for z/OS to create the trace logger, and
3. Code trace requests at appropriate trace points in your application.

Specific instructions for updating your application to use JRas support appear in "Steps for coding your Java application to issue messages and trace requests" on page 289. Before you can use those instructions to properly code trace requests, however, you need to understand the concepts in the following topics:
- "Determining where to place trace points and what data to request"
- "Assigning trace types to trace points"

Determining where to place trace points and what data to request

To collect trace data for a Java application running in a WebSphere for z/OS J2EE or MOFW server, you must decide where to locate trace points in your application's code. At those trace points, you can use `RASTraceLogger` class interfaces to request a trace entry. Typical trace points include:
- Method entry
- Method exit
- Start of a functional request
- Major checkpoints in the process of completing a request
- Completion of a functional request
- Interface to another system function
- Any unusual event, such as a detected I/O error or an unexpected exception

You must also decide what information to record in the trace entries, which can hold a variable amount of data. WebSphere for z/OS automatically collects the address space identifier (ASID) and task control block (TCB) for the unit of work or transaction, and Java name for the thread. The following are suggestions on the additional types of data you might place in the trace entries for a Java application running in a WebSphere for z/OS J2EE or MOFW server:
- Identification of the unit of work or transaction that is being serviced by the application. This can be the JOBNAME, USERID, or transaction identifier.
- For entries that trace the start of a functional request, the input parameters.
- For internal checkpoints, an identification that ties this trace entry to the original request, and information on the current status of the process.
- For unusual events, the cause of the problem and any additional data. For example, you could record any exceptions and stack traces.
- On return from a service, the return code and reason code.
- For trace entries being used for analysis rather than as a debugging aid, whatever information the user of the application needs.

Assigning trace types to trace points

For each trace point you define in your Java application, you use methods defined in the `RASITraceLogger` interface to request trace entries. As part of each trace request, you should assign a trace type for this specific request. The `RASITraceEvent` interface defines the types that you may use.

Note: The Enterprise Edition V3.02 JRas support required you to assign a trace level to trace points in your application. These assignments are still supported, so you do not have to recode any applications that use trace levels.

After you code trace requests, your Java application is capable of issuing trace requests while it runs. To actually record the trace data requested, however, the WebSphere for z/OS J2EE or MOFW server in which your application runs must be enabled for tracing. "Steps for preparing the z/OS or OS/390 environment for logging Java application messages and trace requests" on page 294 provides more detail about enabling tracing for specific trace types.

Steps for coding your Java application to issue messages and trace requests

By coding instructions for issuing messages and logging trace entries, you can improve the reliability, availability, and serviceability (Ras) of your Java server application. When your Java application runs in a WebSphere for z/OS J2EE or MOFW server, its messages appear in one or more of the following destinations, depending on the message type:

- The z/OS or OS/390 master console
- The error log stream that WebSphere for z/OS uses
- The component trace (CTRACE) data set that WebSphere for z/OS uses.

The application's trace entries appear in the same CTRACE data set.

Before you begin:

- If you want to issue messages from your Java application, you may either define the messages inline, or use a separate file to contain the messages. Decide which approach you want to use before you start coding. If necessary, see "Defining messages through inline method calls or a message properties file" on page 286 for more information about these two approaches.
- For descriptions of the JRas interfaces and methods you can use to issue messages or log trace entries, refer to the InfoCenter at `http://www.ibm.com/software/webservers/appserv/library.html`. The InfoCenter describes the JRas Facility methods in the `com.ibm.ras` package, as it applies to all supported platforms, including z/OS or OS/390.

Perform the following steps to add code to your Java server application to direct messages and trace entry requests to z/OS or OS/390 message and trace data logging facilities.

1. (Optional) Create a message properties file if you want to log messages from your application, and have not defined messages inline. For each message that the Java application issues, define the message in a key/text pair:
   - Use the text portion to indicate what is to appear on the master console or in the error log stream
   - Use the key, in both the message properties file and in your Java application code, to enable the run-time code to find the correct message text.

   Rules:
   - Always use an equals sign to separate the key from the text. For example:
     ```
     BBOJ0001=BBOJ0001 Java BO created.
     BBOJ0002=BBOJ0002 Policy number {0} obtained.
     ```
   - Message text that contains variable data requires special coding to indicate the placement and content. To correctly define messages with variable text, use braces {} to indicate that a variable is to appear at a particular place in the text. Within the braces, use a digit to indicate which variable belongs at this place.

     For example, suppose your code contains the following instructions:
     ```
     String day = "Monday";
     Integer temp = new Integer(75);
     msgLogger.message(RASIMessageEvent.TYPE_INFO,
                       this,
     ```

```
                "methodName",
                "APPL061I",
                day,
                temp);
```

To correctly define this message, you would code the following in your message properties file:

```
APPL061I=APPL061I On {0}, it is {1} degrees.
```

---

2. Using an appropriate application development tool for your application, edit the source code for your Java application as follows:
   - Add import statements for the `com.ibm.ras` and `com.ibm.WebSphere` packages. For example, type the following:
     ```
     import com.ibm.ras.*;
     import com.ibm.websphere.ras.*;
     ```
   - Add definition statements for the message and trace loggers. For example, type the following:
     ```
     private RASIMessageLogger msgLogger = null;
     private RASITraceLogger trcLogger = null;
     ```

---

3. Edit the constructor for your Java application to create the message logger, trace logger, or both:

For this type of logger:	Complete the following steps:
Message	• Use the createRASMessageLogger method to request a message logger • (Optional) Define the message properties file, if you are using the file, rather than inline, approach for issuing messages from your application.
Trace	Use the createRASTraceLogger method to request a trace logger

   Rules:
   - Applications must refer to the object returned by the `createRASMessageLogger` method as a type RASIMessageLogger object.
   - Applications must refer to the object returned by the `createRASTraceLogger` method as a type RASITraceLogger object.

   Tip: Avoid using logger names that begin with the `com.ibm.` prefix, which is reserved for use by WebSphere for z/OS.

---

4. If you want to issue messages from your Java application, add messages at appropriate points in the application's source code.

   Rules:
   - If you are defining messages inline, use the `textMessage` methods in the `RASIMessageLogger` interface, specifying the complete message in a string on the method call.
   - If you are using a message properties file, use the `message` or `msg` methods in the `RASIMessageLogger` interface, specifying the message key on the method call. For example:

```
msgLogger.message(RASIMessageEvent.TYPE_INFO,
                  "com.myCompany.JRasSample",
                  "doSomething",
                  "BBOJ0001");
```

- For each message, assign an appropriate type, as defined in the `RASIMessageEvent` interface. These types define the destination of each message:

Message type	Destination
`TYPE_INFORMATION` or `TYPE_INFO`	Master console and CTRACE data set
`TYPE_ERROR` or `TYPE_ERR`	Error log and CTRACE data set
`TYPE_WARNING` or `TYPE_WARN`	CTRACE data set only

Notes:

a. Assign only one message type to each message.
b. If you do not assign a type to a message, or specify ″null″ for the type, the Java compiler issues an error message.
c. If you assign a type that is not valid, the message logger processes the message as a `TYPE_INFORMATION` (or `TYPE_INFO`) message.

- Each character used in a message must map to an EBCDIC character.
- When routing a message to the master console, WebSphere for z/OS sends only the first 700 characters of message text.

Limitation: When writing an error message to the error log stream, WebSphere for z/OS uses only 512 characters of data, including the information it adds to the message text for identification. (This additional information consists of the date, time, organization name, and so on.) See *WebSphere Application Server V4.0.1 for z/OS and OS/390: Messages and Diagnosis*, GA22-7837 for the format and content of error log stream entries for application messages.

---

5. If you want to collect trace data for your Java application, add trace requests at appropriate points in the application's source code.

Rules:

- For each trace request, assign an appropriate type as defined in the RASITraceEvent interface.

  Note: If you do not assign a type to a trace request, the trace logger ignores that trace request.
- Each character used in trace data must map to an EBCDIC character.

Limitation: When processing trace data, WebSphere for z/OS uses only a limited amount of hexadecimal or character data:

- For hexadecimal trace data (from tracing Java byte arrays), WebSphere for z/OS truncates the data after 1024 bytes.
- For character trace data, WebSphere for z/OS substitutes the literal `***BUFFER OVERFLOW***` when that trace data exceeds 16384 characters. This cumulative limit includes 1-byte string terminators for each character string.

Tip: To improve your application's performance, you may use one of the following alternatives:

- Wrap trace calls in a test of the `RASTraceLogger.isLogging` variable, which is set to `false` when trace logging is not active.
- Use the `isLogging` method in an `if` statement to test whether trace logging is active for any level of tracing.
- Use the `isLoggable` method to determine whether logging is active for the designated trace type.

With the first two approaches, the overhead of creating a trace entry does not take place if trace logging is not active. In contrast, the `isLoggable` method requires more overhead, but might be the better option, especially if some level of tracing is always active.

6. Using the appropriate application development tools for your Java application, generate and compile the code for your application.

When you have executable code for your Java application, you are ready to complete the steps listed in "Steps for preparing the z/OS or OS/390 environment for logging Java application messages and trace requests" on page 294.

Example: The following example illustrates the coding requirements described in the instructions above. The example assumes the use of a message properties file, named `com/myCompany/JRasSample.properties`, which contains the following message definitions:

```
BBOJ0001=BBOJ0001 Java BO created.
BBOJ0002=BBOJ0002 Policy number {0} obtained.
BBOJ0003=BBOJ0003 Java BO destroyed.
```

```
package com.myCompany;

// Import JRas and Websphere packages
import com.ibm.ras.*;
import com.ibm.websphere.ras.*;

public class JRasSample
{
  // Loggers
  private RASIMessageLogger msgLogger = null;
  private RASITraceLogger trcLogger = null;
  // Message file
  private static final String MSG_FILE = "com.myCompany.JRasSample";
  // Array of trace objects
  Object[] objs = new Object[3];

  // Constructor
  public JRasSample()
  {
    // Get logger manager object
    Manager manager = Manager.getManager();
    // Get logger
    trcLogger = manager.createRASTraceLogger("com.myCompany","myProduct",
                                        "myComponent","myLogger.COM");
    msgLogger = manager.createRASMessageLogger("com.myCompany","myProduct",
                                          "myComponent","myLogger.COM");
    msgLogger.setMessageFile(MSG_FILE);
  }

  // Example of JRas trace events and messages
  public int doSomething(String parm1,String parm2,String parm3)
  {
    int returnValue = 0;
    byte[] byteArray = {1,2,3,4,5};

    // Trace input parameters
    objs[0] = parm1;
    objs[1] = parm2;
    objs[2] = parm3;
    if (trcLogger.isLoggable(RASITraceEvent.TYPE_ENTRY_EXIT))
      trcLogger.entry(RASITraceEvent.TYPE_ENTRY_EXIT,
                      "com.myCompany.JRasSample",
                      "doSomething",
                      objs);
    if (trcLogger.isLoggable(RASITraceEvent.TYPE_MISC_DATA))
    {
      // Trace a text string
      trcLogger.trace(RASITraceEvent.TYPE_MISC_DATA,
                      "com.myCompany.JRasSample",
                      "doSomething",
                      "Text data to be traced");
      // Trace binary data
      trcLogger.trace(RASITraceEvent.TYPE_MISC_DATA,
                      "com.myCompany.JRasSample",
                      "doSomething",
                      byteArray);
      // Trace the current stack
      trcLogger.stackTrace(RASITraceEvent.TYPE_MISC_DATA,
                           "com.myCompany.JRasSample",
                           "doSomething");
    }
    // Issue informational message to WTO and CTRACE
    msgLogger.message(RASIMessageEvent.TYPE_INFO,
                      "com.myCompany.JRasSample",
                      "doSomething",
                      "BBOJ0001");
    // Issue warning message to CTRACE
    msgLogger.message(RASIMessageEvent.TYPE_WARN,
                      "com.myCompany.JRasSample",
                      "doSomething",
                      "BBOJ0002",
                      "123");
    // Issue error message to error log and CTRACE
    msgLogger.message(RASIMessageEvent.TYPE_ERR,
                      "com.myCompany.JRasSample",
                      "doSomething",
                      "BBOJ0003");
    // Trace return value
    if (trcLogger.isLoggable(RASITraceEvent.TYPE_ENTRY_EXIT))
      trcLogger.exit(RASITraceEvent.TYPE_ENTRY_EXIT,
                     "com.myCompany.JRasSample",
                     "doSomething",
                     returnValue);
    return returnValue;
  }

  // This method is invoked when a JRasSample object is traced
  public String toString()
  {
    String traceString = "This is the JRasSample object trace data";
    return traceString;
  }

  public static void main(String[] args)
  {
    JRasSample sample = new JRasSample();
    sample.doSomething("parm1","parm2","parm3");
  }
}
```

Steps for preparing the z/OS or OS/390 environment for logging Java application messages and trace requests

Before you begin:

- Check with the appropriate installation personnel to determine whether error log streams and component trace data sets were set up during the installation process for WebSphere for z/OS. While error logs and CTRACE data sets might be available already, your installation personnel might determine that changes are necessary to handle your application data, as well as current data from other WebSphere for z/OS servers and applications. For example, your installation may set up either a common error log stream for all WebSphere for z/OS servers, or a separate log stream for each individual server. Your installation might want to switch from using a common log to separate logs, to accomodate additional diagnostic data from your Java applications.
- To turn on tracing for an application in a J2EE or MOFW server, you need to edit or create a JVM properties file. This task might require special authorization to edit or store this file in the appropriate directory. Check with the system programmer who installed WebSphere for z/OS on your system.

Notes:

1. Instructions for setting up error log streams appear in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834.
2. Instructions for setting up and running CTRACE appear in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Messages and Diagnosis*, GA22-7837.

Perform the following steps to set up the z/OS or OS/390 environment for JRas support:

1. On z/OS or OS/390, create a trace settings file in the hierarchical file system (HFS), if you want to enable the WebSphere for z/OS J2EE or MOFW server to collect and log your application's trace data. In this file, type the trace settings that you want, in the following format: `logger_name=type=[enabled|disabled]`

   Example:

   ```
   myLogger.COM=all=enabled
   ```
   logger_name corresponds to the logger name that you specified in the source code for your application, when you coded the create method to obtain a trace logger. To enable logging support for more than one logger name, you may specify a common prefix with an asterisk (for example, `a.b.c.*`), rather than spelling out each logger name in its entirety. Specifying something like `a.b.c.*` enables logging for loggers named `a.b.c.d` and `a.b.c.e`
   Tip: Avoid using logger names that begin with the `com.ibm.` prefix, which is reserved for use by WebSphere for z/OS.
   type corresponds to one of the property values in the following table. Property types are case-sensitive.

Table 24. Trace setting property types and their corresponding JRas trace types

Specifying this property type:	Enables tracing for the following JRas trace types:
all	All supported RASITraceEvent types
event	• RASITraceEvent.TYPE_ERROR_EXC • RASITraceEvent.TYPE_SVC • RASITraceEvent.TYPE_OBJ_CREATE • RASITraceEvent.TYPE_OBJ_DELETE • RASITraceEvent.TYPE_LEVEL1

Table 24. Trace setting property types and their corresponding JRas trace types (continued)

Specifying this property type:	Enables tracing for the following JRas trace types:
entryExit	• RASITraceEvent.TYPE_ENTRY_EXIT • RASITraceEvent.TYPE_API • RASITraceEvent.TYPE_CALLBACK • RASITraceEvent.TYPE_PRIVATE • RASITraceEvent.TYPE_PUBLIC • RASITraceEvent.TYPE_STATIC • RASITraceEvent.TYPE_LEVEL1 • RASITraceEvent.TYPE_LEVEL2
debug	• RASITraceEvent.TYPE_MISC_DATA • RASITraceEvent.TYPE_LEVEL1 • RASITraceEvent.TYPE_LEVEL2 • RASITraceEvent.TYPE_LEVEL3

Rules:

- You may use the same trace properties file to enable different trace types for given loggers. If you do not use a separate line to define each logger's trace types, you must use a single colon (:) to distinguish each logger's trace settings.

  Example (separate line for each logger):

  ```
  com.aCompany.*=all=enabled
  com.anotherCompany.*=event=enabled
  ```

  Example (same line for each logger):

  ```
  com.aCompany.*=all=enabled:com.anotherCompany.*=event=enabled
  ```

- To specify more than one trace type for a logger, separate each trace type with a comma (,)

  Example:

  ```
  com.aCompany.aComponent=debug=enabled,event=enabled
  ```

---

2. Create a new or edit an existing Java virtual machine (JVM) properties file to point to the trace settings file you just created. This properties file, named `jvm.properties`, changes the default settings for the JVM that runs in a WebSphere for z/OS J2EE or MOFW server.

   Rules:

   - You must set the `com.ibm.ws390.trace.settings` system property to the fully qualified directory path and file name for your trace settings file. If you do not specify this system property, or specify the path and file name incorrectly, all trace types are disabled (the default setting).
   - You must make the `jvm.properties` file accessible to WebSphere for z/OS, so it can find and use your property settings when activating the server. Place the `jvm.properties` file in the same HFS directory in which WebSphere for z/OS places the `current.env` file containing environment variable settings for the server in which your Java application will run. See Appendix A, "Environment and JVM properties files," on page 299 for more information about this directory.
   - Trace logging cannot be dynamically started or stopped.

---

3. Check the environment variable settings related to the J2EE or MOFW server's use of component trace. You might want to modify some of the values to

accomodate additional trace entries in the CTRACE data set. Specifically, check the following environment variable settings:
- TRACEBUFFCOUNT
- TRACEBUFFSIZE

---

4. Start the WebSphere for z/OS J2EE or MOFW server in which your application will run:
   - If you have set up JRas support for an existing application that you already installed in a server, you need to:
     a. Make sure your newly compiled code replaces the existing code.
     b. Make sure the WebSphere for z/OS server picks up any modifications you made to the `jvm.properties` file or the environment variables. You need to stop and restart the server to pick up these changes.
   - If you have set up JRas support for a brand-new application, follow the appropriate process to assemble and install your Java application in a WebSphere for z/OS server. For applications to be installed in a J2EE server, see the information in Part 2, "Creating, assembling and deploying J2EE server applications," on page 107.

Background on viewing messages and trace data

Once your Java application starts running, you can view its messages and trace data, as follows:

If you want to view this type of output:	**Use the following instructions:**
Messages on the z/OS or OS/390 master console	The message logger automatically routes messages to the master console in a readable format. Their appearance and duration depend on how your installation has set up its console configuration. If necessary, see *z/OS MVS Planning: Operations*, SA22-7601 for an explanation of ways to configure consoles, including controlling message display, scrolling, and deletion.
Messages in the error log stream	To view messages in the error log stream, use the log browse utility (BBORBLOG). See *WebSphere Application Server V4.0.1 for z/OS and OS/390: Messages and Diagnosis*, GA22-7837 for instructions for using the log browse utility, and for examples of message output.
Messages or trace data in Component Trace	To view messages or application trace data in Component Trace, you must use the interactive problem control system (IPCS) in one of the following ways: • Line mode on a terminal (IPCS CTRACE command), • Full-screen mode on a terminal (IPCS dialog), or • Batch mode, using the terminal monitor program. **Recommendation:** If you are not familiar with IPCS, TSO/E and ISPF, use IPCS in batch mode to format and view trace data, as described in "Steps for using IPCS in batch mode to format application trace data" on page 297. See *WebSphere Application Server V4.0.1 for z/OS and OS/390: Messages and Diagnosis*, GA22-7837 for instructions for using the IPCS dialog, and for examples of message and trace data output.

Note: When you view the trace data for your Java application, messages and CTRACE records might not appear in the order in which your application issued the message or trace requests. All message requests appear in sequential order, relative to each other. Similarly, all CTRACE records appear in order, relative to each other. Different types of trace data, however, might not be in sequence; for example, messages issued after trace requests might show up in trace output before the trace requests.

Steps for using IPCS in batch mode to format application trace data

To view messages or application trace data from Component Trace, you must use the interactive problem control system (IPCS) to format the data. Using IPCS in batch mode is the easiest method of formatting data, especially if you do not have much experience with using IPCS, TSO/E and ISPF. Through batch mode, you can use IPCS to format trace data and write it to an MVS data set. Optionally, you may copy the contents of that data set into an HFS file for viewing.

Before you begin: You must create an IPCS dump directory before you can use IPCS in batch mode. When setting up IPCS, your installation may customize IPCS for its users. This customization can include modifying the IBM-supplied BLSCDDIR CLIST with default values for creating an IPCS dump directory.

If your installation has modified the BLSCDDIR CLIST, perform the following steps to create an IPCS dump directory:

1. Decide on a fully-qualified data set name for the directory.
2. From the TSO/E command prompt, enter the `BLSCDDIR` command, specifying the data set name. For example, to create a dump directory named IBMUSER.DDIR, enter:
   ```
   %blscddir dsn('ibmuser.ddir')
   ```

If your installation has not customized IPCS, you might need to alter other BLSCDDIR CLIST parameters. See *z/OS MVS IPCS User's Guide*, SA22-7596 and *z/OS MVS IPCS Commands*, SA22-7594 for more details about using the BLSCDDIR CLIST to create a dump directory.

Perform the following steps to use IPCS in batch mode to format application trace data:

1. Create a file and copy the following sample JCL into it. This JCL invokes IPCS to extract and format JRAS trace data and write it into an MVS data set, and then uses the `TSO/E OPUT` command to copy the formatted data from the MVS data set into an HFS file.
   ```
   //IBMUSERX   JOB ,
   // CLASS=J,NOTIFY=&SYSUID,MSGCLASS=H
   //IPCS       EXEC PGM=IKJEFT01,REGION=4096K,DYNAMNBR=50
   //IPCSDDIR   DD DSN=IBMUSER.DDIR,DISP=SHR
   //IPCSDOC    DD SYSOUT=H
   //JRASTRC    DD DSN=IBMUSER.CB390.CTRACE,DISP=SHR
   //IPCSPRNT   DD DSN=IBMUSER.IPCS.OUT,DISP=OLD
   //SYSTSPRT   DD SYSOUT=*
   //SYSTSIN    DD *
   IPCS
   DROPDUMP DDNAME(JRASTRC)
   PROFILE LINESIZE(80)PAGESIZE(99999999)
   SETDEF NOCONFIRM
   CTRACE COMP(SYSBBOSS) DDNAME(JRASTRC) FULL PRINT +
          NOTERMINAL
   DROPDUMP DDNAME(JRASTRC)
   END
   ```

```
/*
//OPUT       EXEC PGM=IKJEFT01,REGION=4096K,DYNAMNBR=50
//SYSTSPRT   DD SYSOUT=*
//SYSTSIN    DD *
oput 'ibmuser.ipcs.out' '/u/ibmuser/ipcs/jrastrace.txt' TEXT
/*
```

---

2. Edit the sample JCL to replace IBMUSER.DDIR with the data set name that you used for the IPCS dump directory you created.

   Notes:

   a. Use the PAGESIZE parameter on the PROFILE statement only if you do not want to print the output data set.
   b. You may replace the HFS file name with the name of an existing HFS file, but you do not have to do so. The OPUT command processing will create a new HFS file, if the one specified does not exist, and grants read and write access to that file for your user ID only.

      If you do specify an existing HFS file, the OPUT command processing will write over any data that is already in that file. If you want to know more about the OPUT command, see *z/OS UNIX System Services Command Reference*, SA22-7802.
   c. Change the data set name specified on the JRASTRC DD in the example to the name of the data set containing the CTRACE data.
   d. Change the name of the MVS data set on both the JRASTRC DD statement and the OPUT command in the SYSTSIN stream, as necessary. The formatted output of the JRAS CTRACE data is first written to the MVS data set specified by the IPCSPRNT DD statement and then (optionally) copied to the HFS data set. You must either pre-allocate this data set, or change the sample JCL to allocate the data set. This data set should have a record format of VBA and a record length of 133.

---

3. Submit the JCL to start the IPCS batch job.

Once you are done you can use a UNIX editor, such as vi, to view your trace data in the HFS file. If you want to know more about the UNIX editors, see *z/OS UNIX System Services User's Guide*, SA22-7801.

To learn about formatting CTRACE data with the IPCS dialog, see *WebSphere Application Server V4.0.1 for z/OS and OS/390: Messages and Diagnosis*, GA22-7837.

Appendix A. Environment and JVM properties files

This appendix provides reference information for:
- Environment files and environment variables
- JVM properties files and properties

Environment files and environment variables

This section describes:
- How WebSphere for z/OS manages environment variables and environment files.
- How run-time server start procedures point to their environment files.
- Environment variables for z/OS or OS/390 clients.
- The syntax and meaning of the run-time environment variables.

How WebSphere for z/OS manages server environment variables and environment files

After the bootstrap process during installation and customization, WebSphere for z/OS manages environment data through the Administration application and writes the environmental data into the system management database. To add or change environment variable data, you must enter environment data pairs (an environment variable name and its value) on the sysplex, server, or server instance properties form. When you activate a conversation or prepare for a cold start, the environment variable data is written to HFS files. WebSphere for z/OS determines which values are the most specific for an environment file. For instance, a setting for a server instance takes precedence over the setting for the same variable for its server, and a setting for a server takes precedence over the setting for the same variable for its sysplex.

If you modify an environment file directly and not through the Administration application, any changes are overwritten when you activate a conversation or prepare for a cold start.

When you activate a conversation or prepare for a cold start, WebSphere for z/OS writes the environment data to an HFS file for each server instance. The path and name for each environment file is:

```
CBCONFIG/controlinfo/envfile/SYSPLEX/SRVNAME/current.env
```

where

CBCONFIG
Is a read/write directory that you specify at installation time as the directory into which WebSphere for z/OS is to write configuration data and environment files. At installation time, we call this directory TARGETDIR. The default is `/WebSphere390/CB390`.

Rule: The System Management group (default CBCFG1) and user ID (default CBSYMSR1) must own each directory and subdirectory in CBCONFIG. If the System Management group and user ID do not own CBCONFIG, use the chown command to make them the owner of each directory and subdirectory in CBCONFIG. Thus, if you use the default CBCONFIG, you must use the chown command to give the System Management group and user ID ownership of /WebSphere390 and /WebSphere390/CB390.

Example:

```
chown -R CBSYMSR1:CBCFG1 /WebSphere390
```

SYSPLEX
Is the name of your sysplex. WebSphere for z/OS derives this name from the predefined &SYSPLEX JCL variable.

SRVNAME
Is the server instance name.

| Except for the initial installation of WebSphere for z/OS, you must manage the
| environment variables through the Administration application. At initial
| installation, the customization dialog modifies an initial environment file, which
| the bootstrap job uses.

There are, therefore, two distinct situations in which you define environmental data for your servers. Matching those situations are two distinct ways you create the environment data:

| 1. Prior to the bootstrap process, the customization dialog creates the environment
| file for you. The bootstrap job reads the file and places the environmental data
| into the system management database.
2. Defining and managing environmental data through the Administration application. In this situation, you enter environment data pairs (an environment name and its value—no "=") through a panel in the Administration application.

How run-time server start procedures point to their environment files

WebSphere for z/OS run-time server start procedures must point to an environment file for configuration information. The start procedures use a BBOENV DD statement with a PATH parameter that points to an HFS file. The BBOENV DD statement is:

```
//BBOENV    DD   PATH='&CBCONFIG/&RELPATH/&SYSPLEX/&SRVNAME/current.env'
```

where

&CBCONFIG
Is a variable you set in the start procedure. It must match the read/write directory that you specify at installation time as the directory into which WebSphere for z/OS is to write configuration data and environment files. The default is `WebSphere390/CB390`.

&RELPATH
Is a subdirectory (`controlinfo/envfile`). Its value must not change.

&SYSPLEX
Is the name of your sysplex. Because it is a predefined JCL variable, you do not need to set it in your start procedure.

&SRVNAME
Is the server instance name. By specifying the server instance name when you start the procedure, you can use the same start procedure for other server instances.

Example: To pass the server instance name BBOASR1A to its start procedure, specify:

```
s bboasr1.bboasr1a,srvname='BBOASR1A'
```

To use the same start procedure for server instance BBOASR1B, specify:

```
s bboasr1.bboasr1b,srvname='BBOASR1B'
```

Environment variables for z/OS or OS/390 clients

The Administration application does not manage environment variables for z/OS or OS/390 clients. You must create and manage z/OS or OS/390 client environment files and point to them from client programs. Table 25 on page 303 tells you which environment variables are required or optional for z/OS or OS/390 clients.

Note on using substitution variables

You cannot use variable substitution ($ variables) in environment statements. The variable substitution that is used in UNIX shell environments is not implemented in the Language Environment (LE). Because WebSphere for z/OS processes environment variables in the Language Environment, use of variables such as $PATH in a path environment variable will fail.

Example:

UNIX shell environments often set up paths by appending the new path to the existing path, like this:

```
PATH=yourdir
PATH=$PATH/mydir
```

The resulting path is PATH=yourdir/mydir after substitution for the $PATH variable. However, because WebSphere for z/OS processes the environment variables in the Language Environment, where no variable assignment is made, the resulting path would be PATH=$PATH/mydir.

Environment variable syntax

You must follow this syntax only when defining your initial environment file before the bootstrap process.

Rules: The following are the syntax rules:

- The syntax of the environment variables follows this pattern:

  ```
  VARIABLE=VALUE
  ```

  Where:

  VARIABLE
  is the environment variable.

  VALUE
  is the setting for the variable. The descriptions define possible values for each variable.

- Leading and trailing white space (blanks or tabs) for both variables and values is ignored.

  Example: The two following lines yield the same result:

  ```
  VARIABLE1=VALUE1
  ```

  and

  ```
     VARIABLE1      =      VALUE1
  ```

- "=" is required.

- Blank lines are ignored.
- Code upper and lowercase characters as documented in this topic.
- To comment out an environment variable, simply add a character, such as ‘#’, to the variable. For example, you could change `TRACEALL=0` to `#TRACEALL=0`. The system ignores such coding because the variable does not begin with an alphabetic character.
- Language Environment limits the size of environment variables to 2K.

Environment variable use

Not all environment variables need to be used for each server or client. Table 25 on page 303 tells you where to use a given environment variable. Here are the meanings for what appears in each column:

- “R” means required.
- “O” means optional.
- “F” means required in a future release.
- A blank in the Default column means the variable is not set.
- A blank in other columns means the variable is not used.

Footnotes appear at the end of the table.

Note: The default settings and examples use the standard _CEE_ENVFILE syntax. You do not use this syntax when defining environmental data in the Administration application.

Table 25. Where to use environment variables

Environment variable=<default>	Daemon server instance	System Management server instance	Naming server instance	Interface Repository instance	J2EE server instance	z/OS or OS/390 client
APP_EXT_DIR= *CBCONFIG*/apps/*SRVNAME*/app					R[19]	
BBOC_HTTP_BACKLOG=10	O	O	O	O	O	
BBOC_HTTPS_BACKLOG=10	O	O	O	O	O	
BBOC_HTTP_IDENTITY=					R[1]	
BBOC_HTTP_INPUT_TIMEOUT=10					R[1]	
BBOC_HTTP_LISTEN_IP_ADDRESS=					R[1]	
BBOC_HTTP_MAX_PERSIST_REQUESTS=50					O	
BBOC_HTTP_MODE=					O	
BBOC_HTTP_OUTPUT_TIMEOUT=120					R[1]	
BBOC_HTTP_OUTPUT_TIMEOUT_RECOVERY=[SERVANT]		O	O	O	O	
BBOC_HTTP_SSL_OUTPUT_TIMEOUT_RECOVERY=[SERVANT]		O	O	O	O	
BBOC_HTTP_PERSISTENT_SESSION_TIMEOUT=30					R[1]	
BBOC_HTTP_PORT=					R[1]	
BBOC_HTTP_SSL_CBIND =					O	
BBOC_HTTP_SSL_IDENTITY=					R[2]	
BBOC_HTTP_SSL_INPUT_TIMEOUT=10					O	
BBOC_HTTP_SSL_LISTEN_IP_ADDRESS=					R[1]	
BBOC_HTTP_SSL_MAX_PERSIST_REQUESTS=50					O	
BBOC_HTTP_SSL_MODE=					O	
BBOC_HTTP_SSL_OUTPUT_TIMEOUT=120					O	
BBOC_HTTP_SSL_PERSISTENT_SESSION_TIMEOUT=30					O	
BBOC_HTTP_SSL_PORT=					R[2]	
BBOC_HTTP_SSL_TRANSACTION_CLASS=					R[2]	
BBOC_HTTP_SSL_V3CIPHERS=					O	
BBOC_HTTP_TRANSACTION_CLASS=					R[1]	
BBOC_HTTPALL_NETWORK_QOS=					O[4]	

Table 25. Where to use environment variables (continued)

Environment variable=<default>	**Daemon server instance**	**System Management server instance**	**Naming server instance**	**Interface Repository instance**	**J2EE server instance**	**z/OS or OS/390 client**
BBOC_HTTPALL_TCLASS_FILE =					O	
BBOC_IIOP_BACKLOG=10	O	O	O	O	O	
BBOC_IIOPSSL_BACKLOG=10	O	O	O	O	O	
BBOC_LOG_RESPONSE_FAILURE=NO	O	O	O	O	O	
BBOC_LOG_RETURN_EXCEPTION=NO	O	O	O	O	O	
BBOC_PROPAGATE_UNKNOWN_SERVICE_CONTEXTS=0	O	O	O	O	O	O
BBODUMP=3	O	O	O	O	O	
BBODUMP_CEE3DMP_OPTIONS=	O	O	O	O	O	
BBOLANG=ENUS	O	O	O	O	O	O
BBOO_ACCEPT_HTTP_WORK_AFTER_MIN_SRS=0					O	
BBOO_ACCEPT_HTTP_WORK_AFTER_N_SECS=*n*					O	
BBOO_ACCEPT_HTTP_WORK_AFTER_N_SRS=*n*					O	
BBOO_WORKLOAD_PROFILE=*value*		O			O	
BEAN_DELETE_SLEEP_TIME=4200		R[5]			O[21]	
CBCONFIG=/WebSphere390/CB390	R	R	R	R	R	
CLASSPATH=		O	O	O	O[6]	
CLIENT_DCE_QOP=NO_PROTECTION						O
CLIENT_HOSTNAME=						O
CLIENTLOGSTREAMNAME=						O
CLIENT_RESOLVE_IPNAME=<value for RESOLVE_ IPNAME>		O	O	O	O	O
CLIENT_TIMEOUT=						
CLONEID=					O	
com.ibm.ws.naming.ldap.containerdn= <ibm-wsnTree=t1,o=<org>, c=<country>>			O			
com.ibm.ws.naming.ldap.domainname= *domain name*			O			
com.ibm.ws.naming.ldap.masterurl= ldap://<ip name>:<port>			O			

Table 25. Where to use environment variables (continued)

Environment variable=<default>	**Daemon server instance**	**System Management server instance**	**Naming server instance**	**Interface Repository instance**	**J2EE server instance**	**z/OS or OS/390 client**
com.ibm.ws390.server.classloadermode=2					O	
CONFIGURED_SYSTEM=	R[7]	R[7]	R[7]	R[7]	R[7]	
DAEMON_IPNAME=	R	O				
DAEMON_PORT=5555	O[8]	O[8]				
DATASHARING=1	O	O	O	O		
DEFAULT_CLIENT_XML_PATH=						O[9]
DEFAULT_UNAUTH_CLIENT_ID=CBGUEST		O				
DM_GENERIC_SERVER_NAME=CBDAEMON	O[8]	O[8]				
DM_SPECIFIC_SERVER_NAME=DAEMON01	O[10]	O[10]	O[10]	O[10]	O[10]	
ENABLE_TRUSTED_APPLICATIONS=0						R[3]
HOME=						O
IBM_JVM_ST_VERBOSEGC_LOG=		O			O	
IBM_OMGSSL=0					O	
ICU_DATA=/usr/lpp/WebSphere/bin/		R				
IIOP_SERVER_SESSION_KEEPALIVE=*n*	O	O			O[20]	
IR_GENERIC_SERVER_NAME=CBINTFRP		O				
IR_SPECIFIC_SERVER_NAME=INTFRP01	O[10]	O[10]	O[10]	O[10]	O[10]	
IRPROC=BBOIR	O	O				
IVB_DEBUG_ENABLED=					O[11]	O[11]
IVB_DRIVER_PATH= /usr/lpp/WebSphere		R				
IVB_TRACE_HOST=						O[11]
IVB_TRACE_PORT=2102						O[11]
java.naming.security.credentials=<password>		O				
java.naming.security.principal=<userid>		O				
JAVA_COMPILER=					O	O
JAVA_IEEE754=						O[12]

Table 25. Where to use environment variables (continued)

Environment variable=<default>	**Daemon server instance**	**System Management server instance**	**Naming server instance**	**Interface Repository instance**	**J2EE server instance**	**z/OS or OS/390 client**
JVM_BOOTCLASSPATH=		O			O	
JVM_BOOTLIBRARYPATH=		O			O	
JVM_DEBUG=		O			O	
JVM_DEBUG_PORT=					O	O[11]
JVM_ENABLE_CLASS_GC=		O			O	
JVM_ENABLE_VERBOSE_GC=		O			O	
JVM_EXTRA_OPTIONS=					O	
JVM_HEAPSIZE=256					O	
JVM_LOCALREFS=		O			O	
JVM_LOGFILE=					O	O
JVM_MINHEAPSIZE=		O			O	
LDAPBINDPW=		F	R[13]			
LDAPCONF=		F	R[13]			
LDAPHOSTNAME=		F	R[13]			
LDAPIRBINDPW=		F		R[14]		
LDAPIRCONF=		F		R[14]		
LDAPIRHOSTNAME=		F		R[14]		
LDAPIRNAME=		F		R[14]		
LDAPIRROOT=		F		R		
LDAPNAME=		F	R[13]			
LDAPROOT=		F	R			
LIBPATH=		O	O	O	O[6]	
LOGSTREAMNAME=	O	O				
MAX_SRS=0					O	
MIN_SRS=[0 for MOFW, 1 for J2EE]					O	
NM_GENERIC_SERVER_NAME=CBNAMING		O				

Table 25. Where to use environment variables (continued)

Environment variable=<default>	**Daemon server instance**	**System Management server instance**	**Naming server instance**	**Interface Repository instance**	**J2EE server instance**	**z/OS or OS/390 client**
NM_SPECIFIC_SERVER_NAME=NAMING01	O[10]	O[10]	O[10]	O[10]	O[10]	
NMPROC=BBONM	O	O				
OTS_DEFAULT_TIMEOUT=30	O	O	O	O	O	
OTS_MAXIMUM_TIMEOUT=60	O	O	O	O	O	
PATH=					O	O
RAS_MINORCODEDEFAULT= NODIAGNOSTICDATA						
RECOVERY_TIMEOUT=15		O	O	O	O	
RECYCLE_J2EE_SERVERS=Y		O	O	O	O	
REM_DCEPASSWORD=						O
REM_DCEPRINCIPAL=						O
REM_PASSWORD=		O[15]	O[15]	O[15]	O[15]	O
REM_USERID=		O[15]	O[15]	O[15]	O[15]	O
RESOLVE_IPNAME=		O[16]	O[17]	O[17]	O[17]	R[18]
RESOLVE_PORT=900		O	O	O	O	O
SESSION_COOKIE_NAME=					O	
SM_DEFAULT_ADMIN= CBADMIN		O				
SM_GENERIC_SERVER_NAME=CBSYSMGT		O				
SM_SPECIFIC_SERVER_NAME=SYSMGT01	O[10]	O[10]	O[10]	O[10]	O[10]	
SMPROC=BBOSMS	O	O				
SOMOOSQL=					O	
SRVIPADDR=	O	O	O	O	O	
SSL_HANDSHAKE_THREAD_COUNT=3		O	O	O	O	
SSL_KEYRING=						O
SSL_SERVER_V3CIPHERS=	R	O	O	O	O	
SYS_DB2_SUB_SYSTEM_NAME=DB2	R	R	R	R	R	
TRACEALL=1	O	O	O	O	O	O

Table 25. Where to use environment variables (continued)

Environment variable=<default>	**Daemon server instance**	**System Management server instance**	**Naming server instance**	**Interface Repository instance**	**J2EE server instance**	**z/OS or OS/390 client**
TRACEBASIC=	O	O	O	O	O	O
TRACEBUFFCOUNT=4	O	O	O	O	O	
TRACEBUFFLOC=(Server: BUFFER, Client: SYSPRINT)	O	O	O	O	O	O
TRACEBUFFSIZE=1M	O	O	O	O	O	
TRACEDETAIL=	O	O	O	O	O	O
TRACEMINORCODE=						
TRACEPARM=00	O					
TRACESPECIFIC=	O	O	O	O	O	O
WAS_JAVA_OPTIONS=	O	O	O	O	O	
WS_EXT_DIRS=					O	

Table 25. Where to use environment variables (continued)

Environment variable=<default>	Daemon server instance	System Management server instance	Naming server instance	Interface Repository instance	J2EE server instance	z/OS or OS/390 client

Notes:

1. Required if using the HTTP Transport Handler to handle HTTP protocol requests to the J2EE server.
2. Required if using the HTTPS Transport Handler to handle HTTPS requests to the J2EE server.
3. Required if using single sign-on capability, Form Based authentication, or a trust association interceptor.
4. Must be running on z/OS Version 1 Release 2 or higher for this environment variable to have any affect. It will be ignored for z/OS Release 1 or OS/390 releases.
5. Required when stateful session beans in J2EE servers are activated based on a transaction, rather than activated only once.
6. Required for server regions that use Java, including the IMS PAA and CICS PAA.
7. This environment variable is automatically added to each server instance's environment file and should not be edited.
8. If you specify a value for the Daemon Server, you must provide the same value for the System Management Server control region.
9. Required when the client uses the System Management Scripting API.
10. You must specify this for the second and subsequent systems in a sysplex.
11. Required only when you are using the IBM Object Level Trace and Distributed Debugger Tools to trace and/or debug client and server application components.
12. Required for Java clients that run on z/OS or OS/390.
13. LDAPCONF is mutually exclusive with LDAPBINDPW, LDAPHOSTNAME, and LDAPNAME. Either LDAPCONF is required, or LDAPBINDPW, LDAPHOSTNAME, and LDAPNAME are required.
14. LDAPIRCONF is mutually exclusive with LDAPIRBINDPW, LDAPIRHOSTNAME, and LDAPIRNAME. Either LDAPIRCONF is required, or LDAPIRBINDPW, LDAPIRHOSTNAME, and LDAPIRNAME are required.
15. Used when a server becomes a remote client of another server.
16. For the control region, the default is the value of DAEMON_IPNAME during bootstrap.
17. For the server region, the default is the local system IP name. Generally, do not code.
18. Optional if a Daemon Server is on the same system as the client, in which case the default is the local system IP name.
19. Required if common JAR files and directories are going to be accessed by multiple applications running in the same J2EE server instance.
20. Control region only.
21. If you have an application that uses large numbers of "activate once" stateful session beans, taking the default for BEAN_DELETE_SLEEP_TIME could cause Java out of memory errors.

Environment variable descriptions

APP_EXT_DIR=*path*

Specifies a directory that can be accessed by multiple applications running in the same J2EE server instance. Classes in JAR or zip files in this directory are loaded into the WebSphere for z/OS run-time by the Application Extension class loader.

Note: See the related information about WebSphere for z/OS class loaders and application modules in "Overview of WebSphere for z/OS classloader operation" on page 120.

Default:

`*CBCONFIG*/apps/*SRVNAME*/app`

where

CBCONFIG

Is a read/write directory that you specify at installation time as the directory into which WebSphere for z/OS is to write configuration data and environment files. The default is `/WebSphere390/CB390`.

SRVNAME

Is the generic server name.

Example: `APP_EXT_DIR=/tmp/ws_com_apps`

BBOC_HTTP_BACKLOG=*n*

An integer value that indicates the maximum queue length for pending connections that use HTTP. You may set the maximum value up to 2147483647, but the specification of the SOMAXCONN statement in the TCP/IP profile may result in limitations to this. The default is 10.

Example:

```
BBOC_HTTP_BACKLOG=25
```

BBOC_HTTPS_BACKLOG=*n*

An integer value that indicates the maximum queue length for pending connections that use HTTPS. You may set the maximum value up to 2147483647, but the specification of the SOMAXCONN statement in the TCP/IP profile may result in limitations to this. The default is 10.

Example:

```
BBOC_HTTPS_BACKLOG=25
```

BBOC_HTTP_IDENTITY=*USER_ID*

Specifies a valid SAF user ID which will be used as the current security principal for this HTTP request. The user ID will be treated as an authenticated user by the Web container. If this variable is not specified, the request will be executed under the server region's identity.

Example:

```
BBOC_HTTP_IDENTITY=SECUR001
```

BBOC_HTTP_INPUT_TIMEOUT=*n*

Sets a time, in seconds, that the J2EE server will wait for the complete HTTP request to arrive after the connection has been established before cancelling the connection. The default value is 10 seconds. Specifying a value of zero disables the time-out function.

Example:

```
BBOC_HTTP_INPUT_TIMEOUT=10
```

BBOC_HTTP_LISTEN_IP_ADDRESS=*IP_ADDRESS*
Specifies the IP address, in dotted decimal format, that WebSphere for z/OS J2EE servers use to listen for HTTP client connection requests. This environment variable is used to specify a specific IP address over which the J2EE server is to receive requests. It causes the server to bind to this specific a specific IP address rather than to the default, which is to bind to all addresses (inaddrany). Normally, the server will listen on all IP addresses configured to the local TCP/IP stack. However, if you want to fence the work or allow multiple heterogeneous servers to listen on the same port, you can use BBOC_HTTP_LISTEN_IP_ADDRESS. The specified IP address becomes the only IP address over which this control region receives inbound HTTP requests.

Example:

```
BBOC_HTTP_LISTEN_IP_ADDRESS=9.117.43.16
```

BBOC_HTTP_MAX_PERSIST_REQUESTS=*n*
An integer value indicating the maximum number of HTTP requests that will be processed over a single connection from an HTTP client. When the maximum number of requests have been processed, the client connection will be closed. Set this value to 0 or 1 to turn off persistent connection processing. The default value is 50.

Example:

```
BBOC_HTTP_MAX_PERSIST_REQUESTS=50
```

Note: This environment variable is a replacement for environment variable BBOC_HTTP_SESSION_GC. Once you add the BBOC_HTTP_MAX_PERSIST_REQUESTS environment variable to your current.env file, any value specified for the BBOC_HTTP_SESSION_GC environment variable will be ignored. Therefore, if you have already added environment variable BBOC_HTTP_SESSION_GC to your current.env file, you should delete it.

BBOC_HTTP_MODE=INTERNAL
Indicates that Private Headers received from a WebSphere plug-in for Web servers, over the port specified on the `BBOC_HTTP_PORT` environment variable, are to be trusted. There is no default value for this property. If this property is not included in the current.env file, or if it has a value other than INTERNAL, all Private Headers received over this port will be ignored.

Example:

```
BBOC_HTTP_MODE=INTERNAL
```

BBOC_HTTP_OUTPUT_TIMEOUT=*n*
The time, in seconds, that the J2EE server will wait for the response from the application once the HTTP request has been completed. If the response is not received within the specified length of time, the server region will fail with ABENDEC3 and RC=04130001. The default value is 120 seconds.

Example:

```
BBOC_HTTP_OUTPUT_TIMEOUT=120
```

BBOC_HTTP_OUTPUT_TIMEOUT_RECOVERY=[SESSION|SERVANT]
Controls the recovery action taken on timeouts for requests received over the HTTP transport.

Specifying ″SERVANT″ allows for the termination of server regions when timeouts occur. If an HTTP request is under dispatch in a server region when its timeout value is reached, the server region terminates with an ABENDEC3 RSN=04130001. The HTTP request and socket are then cleaned up.

A setting of ″SESSION″ only cleans up the HTTP request and socket. No attempt is made to disrupt the execution of a dispatched HTTP request within a server region. Be careful using this setting as it may lead to a loss of resources if the dispatched HTTP request loops or hangs.

The default value is ″SERVANT.″

Example:

```
BBOC_HTTP_OUTPUT_TIMEOUT_RECOVERY=SERVANT
```

BBOC_HTTP_SSL_OUTPUT_TIMEOUT_RECOVERY=[SESSION|SERVANT]
Controls the recovery action taken on timeouts for requests received over the HTTP SSL transport.

Specifying ″SERVANT″ allows for the termination of server regions when timeouts occur. If an HTTP SSL request is under dispatch in a server region when its timeout value is reached, the server region terminates with an ABENDEC3 RSN=04130001. The HTTP SSL request and socket are then cleaned up.

A setting of ″SESSION″ only cleans up the HTTP SSL request and socket. No attempt is made to disrupt the execution of a dispatched HTTP SSL request within a server region. Be careful using this setting as it may lead to a loss of resources if the dispatched HTTP SSL request loops or hangs.

The default value is ″SERVANT.″

Example:

```
BBOC_HTTP_SSL_OUTPUT_TIMEOUT_RECOVERY=SESSION
```

BBOC_HTTP_PERSISTENT_SESSION_TIMEOUT=*n*
Specifies the time, in seconds, that the J2EE server will wait for a subsequent request from an HTTP client on a persistent connection. If another request is not received from the same client within this time limit, the connection is closed. The default value is 30 seconds.

Example:

```
BBOC_PERSISTENT_SESSION_TIMEOUT=30
```

BBOC_HTTP_PORT=*n*
Specifies the port at which the J2EE server listens for HTTP requests. Any requests received over the HTTP port will be directed to the Web container for processing.

If this variable is not specified, the J2EE server will not listen for HTTP requests directly.

The use of this HTTP port does not preclude the use of the WebSphere for z/OS plug-in with this J2EE server instance. The Web container is capable of simultaneously processing requests received directly through the HTTP port as well as from the WebSphere for z/OS plug-in.

Note: Currently, HTTP requests received over this HTTP port are not able to be authenticated using the mechanisms described in the J2EE Specification.

Example:

```
BBOC_HTTP_PORT=8080
```

BBOC_HTTP_SSL_CBIND=ON|OFF
If this environment variable is set to ON, all SSL connections from a browser must have a client certificate, and the user ID associated with that client certificate must have RACF CONTROL authority for CB.BIND.servername. If these conditions are not met, the connection will be closed. Issue the following RACF command to give the user ID associated with that client certificate RACF CONTROL authority:

```
PERMIT CB.BIND.servername CLASS(CBIND) ID(clientCertUserid) ACCESS(CONTROL)
```

Example:

```
BBOC_HTTP_SSL_CBIND=OFF
```

BBOC_HTTP_SSL_IDENTITY=*USER_ID*
Specifies a valid SAF user ID which will be used as the current security principal for this HTTPS request. The user ID specified on this variable will be used as the default user ID for the current security principal if no other mechanism is available for establishing client identity (such as a client supplied user ID and password).

The user ID will be treated as an authenticated user by the Web container. If this variable is not specified, the remote identity specified when the J2EE server was configured will be used as the current security principal. (See *WebSphere Application Server 4.0.1 for z/OS and OS/390: System Management User Interface* for more information about specifying the Remote Identity.)

Example:

```
BBOC_HTTP_SSL_IDENTITY=CBGUEST
```

BBOC_HTTP_SSL_INPUT_TIMEOUT=*n*
The time in seconds that the J2EE server will allow for the complete HTTPS request to be received before cancelling the connection. The default value is 10 seconds.

Example:

```
BBOC_HTTP_SSL_INPUT_TIMEOUT=10
```

BBOC_HTTP_SSL_LISTEN_IP_ADDRESS=*IP_ADDRESS*
Specifies the IP address, in dotted decimal format, that WebSphere for z/OS J2EE servers use to listen for HTTPS client connection requests. This IP address is used by the server to bind to TCP/IP. Normally, the server will listen on all IP addresses configured to the local TCP/IP stack. However, if you want to fence the work or allow multiple heterogeneous servers to listen on the same port, you can use BBOC_HTTP_SSL_LISTEN_IP_ADDRESS. The specified IP address becomes the only IP address over which this control region receives inbound HTTPS requests.

Example:

```
BBOC_HTTP_SSL_LISTEN_IP_ADDRESS=9.117.43.16
```

BBOC_HTTP_SSL_MAX_PERSIST_REQUESTS=*n*
An integer value indicating the maximum number of HTTPS requests that will be processed over a single connection from an HTTPS client. When the maximum number of requests have been processed, the client connection will be closed. Set this value to 0 or 1 to turn off persistent connection processing. The default value is 50.

Example:

```
BBOC_HTTP_SSL_MAX_PERSIST_REQUESTS=50
```

BBOC_HTTP_SSL_MODE=INTERNAL
Indicates the Private Headers received from a WebSphere plug-in for Web servers, over the port specified on the `BBOC_HTTP_SSL_PORT` environment variable, are to be trusted. There is no default value for this property. If this property is not included in the current.env file or if it has a value other than INTERNAL, all Private Headers received over this port will be ignored.

Example:

```
BBOC_HTTP_SSL_MODE=INTERNAL
```

BBOC_HTTP_SSL_OUTPUT_TIMEOUT=*n*
The time, in seconds, that the J2EE server will wait for the response from the application once the HTTPS request has been completed. If the response is not received within the specified length of time, the server region will fail with ABENDEC3 and RC=04130001. The default value is 120 seconds.

Example:

```
BBOC_HTTP_SSL_OUTPUT_TIMEOUT=120
```

BBOC_HTTP_SSL_PERSISTENT_SESSION_TIMEOUT=*n*
Specifies the time, in seconds, that the J2EE server will wait between requests issued over a persistent connection from an HTTPS client. After the server sends a response, it uses the persistent timeout to determine how long it should wait for a subsequent request before cancelling the persistent connection. The default value is 30 seconds.

Example:

```
BBOC_HTTP_SSL_PERSISTENT_SESSION_TIMEOUT=30
```

BBOC_HTTP_SSL_PORT=*n*
Specifies the port at which the J2EE server listens for HTTPS requests. Any requests received over the HTTPS port will be directed to the Web container for processing.

If this variable is not specified, the J2EE server will not listen for HTTPS requests directly.

The use of this HTTPS port does not preclude the use of an IBM HTTP Server for z/OS with this J2EE server instance. The Web container is capable of simultaneously processing requests received directly through the HTTPS port as well as from an IBM HTTP Server for z/OS.

Example:

```
BBOC_HTTP_SSL_PORT=8080
```

BBOC_HTTP_SSL_TRANSACTION_CLASS=*TRANSACTION_CLASS*
A valid WLM transaction class, which will be used in the creation of the WLM enclave for all HTTPS requests. If a valid WLM transaction class is not specified, no transaction class will be set for the enclave.

Example:

```
BBOC_HTTP_SSL_TRANSACTION_CLASS=TCLASSA
```

BBOC_HTTP_SSL_V3CIPHERS=*string*
Defines the SSL Version 3 cipher suites that system SSL uses in the SSL handshake for an HTTP SSL connection. It overrides any server-wide setting set via the Administration Application or SSL_SERVER_V3CIPHERS. Specify a string as documented in ″z/OS System Secure Sockets Layer Programming″ (SC24-5901). Each cipher is represented by two characters (for example, ″09″ instead of ″9″). You can specify the string with or without comma delineation.

If you delineate with commas, a validity check will run against the installed ciphers. The default is an empty string, meaning no change is made to the cipher suites.

Examples:

```
BBOC_HTTP_SSL_V3CIPHERS=09,0A,05
BBOC_HTTP_SSL_V3CIPHERS=090A05
```

BBOC_HTTP_TRANSACTION_CLASS=*TRANSACTION_CLASS*
A valid WLM transaction class, which will be used in the creation of the WLM enclave for all HTTP requests. If a valid WLM transaction class is not specified, no transaction class will be set for the enclave.

Example:

```
BBOC_HTTP_TRANSACTION_CLASS=TCLASSA
```

BBOC_HTTPALL_NETWORK_QOS=HOST|URI|HOSTURI|TCLASS
Specifies the parameters that will be used to classify outbound data that is delivered in response to HTTP and HTTPS requests. The classification parameters and values can be used to construct a network Quality of Service (QOS) policy. This environment variable is only effective if you are running WebSphere for z/OS on z/OS Version 1 Release 2 or higher. It will be ignored for lower releases.

For more information about setting a QOS policy, see the *z/OS Communications Server IP Configuration Guide* at URL:

`http://www.ibm.com/servers/eserver/zseries/zos/bkserv/`
If valid values are not provided for this environment variable or if this environment variable is not specified, the response data will not be classified to the network agent. The following parameters can be specified for this environment variable:

HOST
If this parameter is specified, WebSphere for z/OS will classify the outbound response data using the value that was provided in the host header of the request. This value will typically be the domain name by which this response will be exposed to Web clients in a DNS. The port number will be included.

Example:

```
www.mycompany.com
```

URI
If this parameter is specified, WebSphere for z/OS will use the value of the URI in the request line of the request to classify the outbound response data. Any query string will be truncated.

Example:

```
/mywebap/myservlet
```

HOSTURI
If this parameter is specified, WebSphere for z/OS will classify the outbound response data using the values specified for the HOST and URI parameters concatenated together.

Example:

```
www.mycompany.com/mywebap/myservlet
```

TCLASS
If this parameter is specified, WebSphere for z/OS will use the resultant transaction class value that was used to classify the inbound request to the

Z/OS Workload Manager. See *z/OS V1R3.0 MVS Workload Management Services*, SA22-7619, for information on specifying a transaction class value.

Note: This environment variable is ignored if you are running WebSphere for z/OS on an OS/390 Release 8 system.

BBOC_HTTPALL_TCLASS_FILE =<filename>
Specifies the fully qualified name of the file containing the rules for classifying an HTTP or HTTPS request.

Example:

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

In this example, the content of the file tclass.conf will be used to map requests to a transaction class.
If multiple entries match the request, the first successful match is used. If there are not matching entries, the value specified for the `BBOC_HTTP_TRANSACTION_CLASS` environment variable will be used for a non-SSL request, and the value specified for the `BBOC_HTTP_SSL_TRANSACTION_CLASS` environment variable will be used, for an SSL request. If no value was specified for either the `BBOC_HTTP_TRANSACTION_CLASS` or `BBOC_HTTP_SSL_TRANSACTION_CLASS` environment variable, the enclave will get created without a transaction class value.
Following is the syntax for entries in this file:

```
TransClassMap <host>:<port> <uritemplate> <tclass>
```

where:

<host>
Is the value compared against the hostname of the HOST: header of the request. This value can be a wildcard '*'.

Note: A value of '*' for the host:port value is acceptable and is equivalent to '*:*'.

<port>
Is the value compared against the port of the request. This value can be a wildcard '*'.

<uritemplate>
Is the value compared against the URI of the request. Any query string will not be used in the comparison. This value can be a wildcard '*', or end in a wildcard.

<tclass>
Is the Workload Manager Transaction Class name that will be used in the creation of the enclave.

Examples:

```
TransClassMap www.ibm.com:80 /webap1/myservlet TCLASS1
TransClassMap www.ibm.com:* /webap1/myservlet TCLASS2
TransClassMap *:443 * TCLASS3
TransClassMap *:* /webap1/myservlet TCLASS4
TransClassMap www.ibm.com:* /webap2/* TCLASS5
TransClassMap * /myservlet TCLASS6
TransClassMap * * TCLASS6
```

| **BBOC_IIOP_BACKLOG=*n***
| An integer value that indicates the maximum queue length for pending
| connections that use IIOP. You may set the maximum value up to 2147483647,

but the specification of the SOMAXCONN statement in the TCP/IP profile may result in limitations to this. The default is 10.

Example:

```
BBOC_IIOP_BACKLOG=25
```

BBOC_IIOPSSL_BACKLOG=*n*
An integer value that indicates the maximum queue length for pending connections that use IIOP SSL. You may set the maximum value up to 2147483647, but the specification of the SOMAXCONN statement in the TCP/IP profile may result in limitations to this. The default is 10.

Example:

```
BBOC_IIOPSSL_BACKLOG=25
```

BBOC_LOG_RESPONSE_FAILURE=[YES|NO]
Determines whether message BBOU0733W is issued to record a failure detected when attempting to send a response to a client. The message is sent to the error log. YES causes the message to be issued. The default is NO.

The message text will contain the request method name, the reply status, and routing information identifying the client.

Example:

```
BBOC_LOG_RESPONSE_FAILURE=YES
```

BBOC_LOG_RETURN_EXCEPTION=[YES|NO]
Determines whether message BBOU0734W is issued to record a response that contains an SystemException. The message is sent to the error log. YES causes the message to be issued. The default is NO.

The message text will contain the exception identifier and minor code, the request method name, and routing information identifying the client.

Example:

```
BBOC_LOG_RETURN_EXCEPTION=YES
```

BBOC_PROPAGATE_UNKNOWN_SERVICE_CONTEXTS=[0|1]
Activates or deactivates the support for unknown IIOP Service Contexts.

Unknown IIOP Service Contexts received on requests are propagated with the Request. If the dispatched Request invokes an outbound Request, the current set of unknown IIOP Service Contexts are propagated with the new outbound Request. Upon receipt of the response to the new outbound Request, any unknown IIOP Service Contexts received are propagated back to the outbound Request. Also, the set of unknown IIOP Service Contexts are merged back from the outbound Request into the dispatched Request. The response for the original inbound Request contains the current set of unknown IIOP Service Contexts.

The default is 0, which instructs the ORB to not propagate unknown IIOP Service Contexts.

Example:

```
BBOC_PROPAGATE_UNKNOWN_SERVICE_CONTEXTS=1
```

BBODUMP=*n*
Specifies the default dump used by the signal handler. Valid values and their meanings are:

0 No dump is generated.

1 A ctrace dump is taken.

2 A cdump dump is taken.

3 A csnap dump is taken.

4 A CEE3DMP dump is taken. CEE3DMP generates a dump of Language Environment and the member language libraries. Sections of the dump are selectively included, depending on dump options specified, either by default or through the BBODUMP_CEE3DMP_OPTIONS environment variable. By default, this value passes THREAD(ALL) BLOCKS to CEE3DMP. You can override the default options for CEE3DMP through the BBODUMP_CEE3DMP_OPTIONS environment variable.

For more information about CEE3DMP and its options, see *z/OS Language Environment Programming Reference*, SA22-7562.

If you do not specify BBODUMP, the default value is 3 (a csnap dump is taken).

Example:

```
BBODUMP=3
```

BBODUMP_CEE3DMP_OPTIONS=*options*

Specifies dump options to be used with a CEE3DMP. This environment variable is used when you specify BBODUMP=4. For an explanation of CEE3DMP and valid dump options, see *z/OS Language Environment Programming Reference*, SA22-7562.

Rule: The maximum length of the option string on this environment variable is 255. If the option string is longer than 255, you receive message BBOU0514W and the CEE3DMP dump options are set to THREAD(ALL) BLOCKS.

Example:

```
BBODUMP_CEE3DMP_OPTIONS=NOTRACEBACK NOFILES
```

BBOLANG=*LANGUAGE*

The name of the WebSphere for z/OS message catalog used. The default is ENUS.

BBOO_ACCEPT_HTTP_WORK_AFTER_MIN_SRS=[0|1]

A value of 1 indicates that the minimum number of Server Regions must be ready for work before HTTP work will be accepted into the Server. The minimum number of Server Regions is specified on the MIN_SRS environment variable. Once the minimum number of Server Regions are ready for work, the HTTP/S transport will start accepting work. The default is 0 (no Server Regions need to be ready for work before work is accepted over the HTTP/S transport).

Example:

```
BBOO_ACCEPT_HTTP_WORK_AFTER_MIN_SRS=1
```

BBOO_ACCEPT_HTTP_WORK_AFTER_N_SECS=*n*

Defines the amount of time, specified in seconds, to wait for the desired number of server regions as defined by environment variable BBOO_ACCEPT_HTTP_WORK_AFTER_N_SRS before accepting work over the HTTP Transport or HTTP SSL protocols. The default is 300 seconds and the minimum value is 10 seconds.

Example:

```
BBOO_ACCEPT_HTTP_WORK_AFTER_N_SECS=300
```

BBOO_ACCEPT_HTTP_WORK_AFTER_N_SRS=*n*
Defines the number of server regions that must be ready for work before HTTP work is accepted into the server. Once the specified number of server regions are ready, the HTTP Transport and HTTP SSL protocols start accepting work. Both the default and the minimum value are 1.

Example:

```
BBOO_ACCEPT_HTTP_WORK_AFTER_N_SRS=1
```

BBOO_WORKLOAD_PROFILE=*value*
Controls workload-pertinent decisions made by the WebSphere for z/OS runtime, such as the number of threads used in the server region. The default value is NORMAL, which is the appropriate value for most applications. Consider using one of the other values when your application requires more threads.

NORMAL
Gives you the thread count dictated by WebSphere for z/OS--either 1 (single-threaded) or 3 (multi-threaded). This value is the default.

IOBOUND
Use IOBOUND if you want more threads in applications that perform I/O-intensive processing on z/OS. The number of threads is calculated based on your number of CPUs.

CPUBOUND
Use CPUBOUND if you want more threads in applications that perform processor-intensive operations on z/OS. The number of threads is calculated based on your number of CPUs, and cannot be less than three.

LONGWAIT
Use LONGWAIT for application processing that involves sending or receiving information across a network.

Example: `BBOO_WORKLOAD_PROFILE=NORMAL`

BEAN_DELETE_SLEEP_TIME=n
The time in seconds allowed before an expired stateful session bean's state is deleted from its backing datastore (DB2). The default time is 4200 seconds (70 minutes). You can increase the time to 2147483 seconds (24.85 days). Recommendation: Do not set this variable less than 300 seconds (5 minutes).

Note: If you change the value of this variable for your application server, you may also need to adjust the bean timeout value for your stateful beans. The default stateful bean timeout is 8 hours, which, when coupled with the BEAN_DELETE_SLEEP_TIME default value, means it could take up to 9 hours and 10 minutes to delete a bean. See *WebSphere Application Server V4.0.1 for z/OS and OS/390: Assembling J2EE Applications*, SA22-7836, for more information about stateful session bean timeout.

Example:

```
BEAN_DELETE_SLEEP_TIME=1000000
```

CBCONFIG=*path*
Specifies a read/write directory in the HFS into which WebSphere for z/OS writes configuration and environment files when a conversation is activated. The &CBCONFIG variable in control and server region start procedures must match this value. In this way, WebSphere for z/OS can find the appropriate environment file for a server when those start procedures are executed. The default is /WebSphere390/CB390.

Example: `CBCONFIG=/WebSphere390/CB390`

Rules:

1. You cannot change the value for CBCONFIG through the Administration application (SM EUI).
2. The System Management group (default CBCFG1) and user ID (default CBSYMSR1) must own each directory and subdirectory in CBCONFIG. If the System Management group and user ID do not own CBCONFIG, use the chown command to make them the owner of each directory and subdirectory in CBCONFIG. Thus, if you use the default CBCONFIG, you must use the chown command to give the System Management group and user ID ownership of /WebSphere390 and /WebSphere390/CB390.

   Example:

   ```
   chown -R CBSYMSR1:CBCFG1 /WebSphere390
   ```

Recommendation: You should not change the value of CBCONFIG except prior to an initial bootstrap or a cold start of WebSphere for z/OS.

CLASSPATH=*path1***:***[path2]***:...**
Specifies Java class files—.jar files and classes.zip files—for use by Java business objects in server regions. Specify your Java business object's .jar files when you use Java business objects. The entire CLASSPATH statement must be on one line only.

Example:

```
CLASSPATH=/usr/lpp/db2/db2710/classes/db2j2classes.zip: . . .
```

CLIENT_DCE_QOP=*value*
The level of DCE message protection used by a local z/OS or OS/390 client to apply to the current transaction flows. Normally, you would set DCE security for an z/OS or OS/390 client that accesses servers on remote systems. Note that the DCE level for a server is set through the Administration application.

When enabled on client and server, DCE authentication offers each proof of the other's legitimacy with a handshake message exchange using DCE's third-party authentication scheme. Once this exchange has taken place, messages can be assigned one of three levels of protection, which are the values of this environment variable:

NO_PROTECTION
DCE assures only that the messages and their replies are from the legitimate sender. This is the default.

INTEGRITY
DCE assures that the message is from the legitimate sender and it has not been modified in any way since the sender sent it.

CONFIDENTIALITY
DCE encrypts the message so that none but the legitimate receiver can read it.

CLIENT_HOSTNAME=
Allows an z/OS or OS/390 client to determine its host IP name when no Daemon is running on the same system. When a client program issues the CBSeriesGlobal::hostName() method, the system checks the CLIENT_HOSTNAME environment variable first and returns this value, if it is set. If the value is not set, the system returns the IP name of the Daemon running on that system, if the Daemon is running. The default value is null.

Example: `CLIENT_HOSTNAME=MYSYS.SYS.COM`

CLIENTLOGSTREAMNAME=*LOG_STREAM_NAME*
The WebSphere for z/OS error log stream to which an z/OS or OS/390 client ORB writes error information.

Example: `CLIENTLOGSTREAMNAME=MY.CLIENT.ERROR.LOG`

CLIENT_RESOLVE_IPNAME=*IP_NAME*
The Internet Protocol name that an z/OS or OS/390 client, or server region acting as a client, uses to access the bootstrap server (that is, when the client or server region invokes the resolve_initial_references method). The default is the value specified by the RESOLVE_IPNAME environment variable, which is the Internet Protocol name associated with the System Management Server (the default bootstrap server). If RESOLVE_IPNAME is not set, the value is the system on which the client or server region is running.

The CLIENT_RESOLVE_IPNAME environment variable allows you to specify a bootstrap server running on a remote system, while other clients use a local bootstrap server defined by the RESOLVE_IPNAME environment variable.

Note: The TCP/IP port number for the CLIENT_RESOLVE_IPNAME is defined by the RESOLVE_PORT environment variable.

The value of CLIENT_RESOLVE_IPNAME can be up to 255 characters.

Example: `CLIENT_RESOLVE_IPNAME=REMHOST`

CLIENT_TIMEOUT=*n*
Sets the time-out value for response from a client method call. Set in the control region, the time is in tenths of seconds (thus, a value of 10 is 1 second). This is the only time-out available for remote method dispatches. Because the sysplex TCP/IP that runs through the coupling facility does not always tell the client when the other end of the socket is gone, you would normally wait indefinitely for a response. `CLIENT_TIMEOUT` ensures that you get a response within the configured time, even if it's a `COMM_FAILURE` exception. The default value is 0 (unlimited), which means no time-out value is set.

Example:

```
CLIENT_TIMEOUT=20
```

CLONEID=<id>
Specifies the cloneID that is used to provide session affinity across WebSphere for z/OS J2EE server instances. The value specified for this environment variable must match a value specified on a <Server CloneID> element in the plugin-cfg.xml file for the Web server plug-in that is being used with WebSphere for z/OS. The default value for this environment variable is created by the Web container based on the name of the J2EE server and the name of the J2EE server instance with which this cloneID is associated, and is of the form <ServerName.ServerInstanceName>.

If you change the default value, you must make sure:

- The new value matches a value specified on a <Server CloneID> element in the plugin-cfg.xml file for the Web server plug-in that you are using.
- The new value is a combination of the following:
  - English alphanumeric characters (uppercase or lowercase A to Z and numbers 0 to 9)
  - Periods (.)
  - Underscores (_)
  - Hyphens (-)

Note: Alphabetic characters are case-sensitive. The case of any alphabetic character specified here must exactly match the case of that character as it is specified on the <Server CloneID> element in the plugin-cfg.xml file.

com.ibm.ws.naming.ldap.containerdn=*ibm-wsnTree=t1,o=org,c=country*
The starting point of WsnName tree. Only the Naming server uses this environment variable. By default, the system expects the value to be `ibm-wsnTree=t1,o=WASNaming,c=us`. If you take the default, delete this environment variable from your environment file.

This value must match the value specified in LDAP initialization file (our sample is bboldif.cb). If you've modified the organization or country in your bboldif.cb file, use the same value on this environment variable. Note that case does not matter in LDAP, though it does matter for the environment variables. The ″o=,c=″ portion must also be specified as a suffix in bboslapd.conf.

Example:

```
suffix   "o=WASNaming,c=us"
```

Tip: The suffix statement appears as:

```
suffix         "<ws_rdn>"
```

in the sample bboslapd.conf we ship.

Example:

```
com.ibm.ws.naming.ldap.containerdn=ibm-wsnTree=t1,o=WASNaming,c=us
```

com.ibm.ws.naming.ldap.domainname=*domain name*
Uniquely identifies the host root and is the basis for partitioning the JNDI global name space. Only the Naming server uses this environment variable. By default, the system expects the value to be the domain name of the sysplex on which Naming Server is running. If you want the default, delete this environment variable from the environment file. If you want a different domain name, specify it.

Example:

```
com.ibm.ws.naming.ldap.domainname=plex1
```

com.ibm.ws.naming.ldap.masterurl=ldap://*IP_name***:***port*
The LDAP Server IP Name and port number. Only the Naming server uses this environment variable. By default, the system expects the IP name to be the same as the system on which the Naming Server runs and the port to be 1389. If your LDAP server is running on a system other then the one the Naming Server runs on or uses a port other than 1389, update this environment variable. Otherwise, delete this environment variable.

Example:

```
com.ibm.ws.naming.ldap.masterurl=ldap://wsldap:1389
```

com.ibm.ws390.server.classloadermode=*number*
Specifies the type and behavior of class loaders that the J2EE server uses to load a class from an application module. This capability supports different approaches to packaging application components for installation in a J2EE server, and influences the search-path order that WebSphere for z/OS class loaders use to find and load classes. For additional information about application packaging guidelines and classloader operation, see "Overview of WebSphere for z/OS classloader operation" on page 120.

Recommendation: For most applications, use the default value (2, application mode).

Valid values for this property are:

0 Specifies module mode.

Recommendation: Use this value only if your applications have Manifest classpath statements in EJB JAR files or in WAR files. Even in this case, IBM recommends that you change this property setting to 2 (application mode).

With module mode, WebSphere for z/OS uses only one classloader per module. Each module (JAR or WAR file) has its own unique classloader. Visibility of other modules in the application is achieved only when Manifest classpath entries are added to a module.

If you specify module mode, you must include all application dependent files in each application's EAR file, or place them in the directory specified through the `APP_EXT_DIR` environment variable. Any JAR or zip files that exist in the directory specified on this environment variable are considered common files and can be accessed by any application running in the same J2EE server.

1 Specifies compatibility mode, which allows compatibility with applications from previous releases of WebSphere for z/OS. In this mode, all EJB module classloaders have visibility of all other EJB module classloaders, and all Web application modules have visibility of the EJB classloaders. The EJB classloaders are searched in the order in which the EJB modules were initialized.

If you specify compatibility mode, you must include all application dependent files in each application's EAR file, or place them in the directory specified through the `APP_EXT_DIR` environment variable. Any JAR or zip files that exist in the directory specified on this environment variable are considered common files and can be accessed by any application running in the same J2EE server.

2 Specifies application mode, which allows all classloaders in a J2EE application to have visibility of other classloaders in the same application. The search order is the same as the order in which the modules are defined in the application.xml for the EAR file.

If you specify application mode, you must include all application dependent files in each application's EAR file, or place them in the directory specified through the `APP_EXT_DIR` environment variable. Any JAR or zip files that exist in the directory specified on this environment variable are considered common files and can be accessed by any application running in the same J2EE server.

3 Specifies server mode, which allows all application classloaders on a J2EE server to have visibility to all other application classloaders in the server. This setting enables classes in one application to be visible to classes in all of the other applications residing on that server. When server mode is specified, common files do not have to be placed in the directory specified through the `APP_EXT_DIR` environment variable.

Default: 2 (application mode)

Example: `com.ibm.ws390.server.classloadermode=1`

CONFIGURED_SYSTEM=*system*

Specifies the name of the system to which the server instance was originally configured. During prepare for cold start, cold start, and server activation, the run time adds this environment variable to each server instance's environment file automatically.

Rule: Do not manually add or change this environment variable at any time, such as:

- In the initial environment file before bootstrap
- Through the Administration application (SM EUI)
- In an existing server environment file.

DAEMON_IPNAME=*IP_NAME*

The Internet Protocol name that the Daemon Server registers with the Domain Name Service (DNS). Any CORBA client communication with WebSphere for z/OS requires this IP name.

You must define the DAEMON_IPNAME environment variable at installation time, before you start the Daemon bootstrap process. Otherwise, WebSphere for z/OS issues an error message and terminates the Daemon.

The bootstrap process sets, among other things, the Daemon IP name in the system management database. After bootstrap, WebSphere for z/OS uses the value in the system management database. It is possible that, after bootstrap, the value of the DAEMON_IPNAME environment variable could change to a value other than what is in the system management database. If this happens, an error message is issued, but the Daemon initializes with the Daemon IP name from the system management database.

To place Daemon server instances in the same host cluster, you must code the same DAEMON_IPNAME value for each server instance.

Rules:

- The value for DAEMON_IPNAME must be a fully-qualified long name.
- The first-level qualifier can be from 1 to 18 characters.
- Once chosen, the port and IP name for the Daemon should not change, since every object reference includes the port and IP name—if you change them, existing objects will no longer be accessible.

Example: `DAEMON_IPNAME=CBQ091.PDL.POK.IBM.COM`

DAEMON_PORT=*n*

The port number at which the Daemon Server listens for requests. The default is 5555. If you specify a value, you must provide the same value for the System Management Server control region.

Example: `DAEMON_PORT=5555`

DATASHARING=[0 | 1]

Specifies whether a WebSphere for z/OS instance shares DB2 resources with one or more other WebSphere for z/OS members (clustered host instances) of the sysplex. The value can be 0 or 1. The default value 1 means data sharing is active. The value 0 means data sharing is not active. In a monoplex, this variable has no effect and can be set to 1 or 0.

Example:

```
DATASHARING=1
```

DEFAULT_CLIENT_XML_PATH=*path*

Specifies the location of a set of XML files that hold default parameter lists used by the System Management Scripting API. You must set this environment variable for clients that use the System Management Scripting API.

IBM provides a set of sample XML files that contain default parameter lists. After installation, these samples reside in `/usr/lpp/WebSphere/samples/smapi`. For information about the XML files and the parameter lists, see *WebSphere Application Server V4.0.1 for z/OS and OS/390: System Management Scripting API*, SA22-7839.

You can override the default behavior of the System Management Scripting API in two ways:

1. Specifying the parameters explicitly in the REXX script that calls the System Management Scripting API. By specifying parameters explicitly, you do not have to modify the XML samples IBM provides. You simply need to code

   ```
   DEFAULT_CLIENT_XML_PATH=/usr/lpp/WebSphere/samples/smapi
   ```

   in your client environment file.
2. Copying the XML files to another directory (the samples IBM provides are read-only), making modifications to the parameter lists, then changing the DEFAULT_CLIENT_XML_PATH to point to the new directory. Making these changes is required only if you want to override permanently the default behavior of the System Management Scripting API.

Example: `DEFAULT_CLIENT_XML_PATH=/usr/lpp/WebSphere/samples/smapi`

DEFAULT_UNAUTH_CLIENT_ID=*user_id*

The default local and remote user ID that the System Management server associates with servers. If you allow unauthenticated client requests on a server, and do not explicitly specify your own local and remote user ID for that server, those requests run under the authority of this user ID.

If you do not define this environment variable, the default local and remote user ID is CBGUEST.

You must define this user ID to z/OS or OS/390 and give it appropriate security authorizations (for example, RACF permissions and LDAP permissions).

This environment variable is used only by the System Management server. Using this environment variable in the environment file for other servers takes no effect. That is, you cannot use this environment variable for other servers to define the default local and remote ID that is used by those servers. Rather, you must define the default through the server properties panel in the Administration application. To do this

- Select the “Allow non-authenticated clients” checkbox. The Administration application supplies the value for the local and remote identity from the value on the DEFAULT_UNAUTH_CLIENT_ID variable (or, if not specified, it supplies CBGUEST).
- Type over the supplied values with your value.

The System Management server uses this environment variable during bootstrap. After bootstrap, you can modify the value only at the sysplex level through the Administration application.

Example: `DEFAULT_UNAUTH_CLIENT_ID=DUDE`

DM_GENERIC_SERVER_NAME=*SERVER_NAME*

The server name for the Daemon Server. The default is CBDAEMON. If you specify a value, you must provide the same value for the System Management Server control region.

Example: `DM_GENERIC_SERVER_NAME=CBDAEMON`

DM_SPECIFIC_SERVER_NAME=*SERVER_INSTANCE_NAME*
A server instance name of the Daemon Server. The default is DAEMON01. You must specify this environment variable for all server instances in the second and subsequent systems in a sysplex.

Example: `DM_SPECIFIC_SERVER_NAME=DAEMON01`

ENABLE_TRUSTED_APPLICATIONS=
Specifies whether or not custom user registries can be used. Setting this variable to 1 enables the custom user registry function. When this function is enabled, RACF checks if the user is authorized, and if the user is authorized, allows the call to complete. If this variable does not exist, or is not set to 1, the user will get an exception.

HOME=*path*
Specifies the home directory. This variable is set automatically from the security product user profile when the user logs in to the UNIX shell.

| **IBM_JVM_ST_VERBOSEGC_LOG=***filename*
| Specifies the HFS file in which garbage-collection output will be logged. Use
| this variable with both of the following:
| • JVM_ENABLE_CLASS_GC=1 to enable garbage collection, and
| • JVM_ENABLE_VERBOSE_GC=1 to view verbose output from the garbage
| collection.

| **Recommendation:** If you also are using the `JVM_LOGFILE` variable to specify an
| HFS file for JVM-related output, **do not** specify the same HFS file for the
| `IBM_JVM_ST_VERBOSEGC_LOG` variable. WebSphere for z/OS will not append data
| to an existing file; instead, the data will be overwritten if both of these
| variables specify the same HFS file.

IBM_OMGSSL=[0 | 1]
Specifies whether only CORBA-compliant security tags will be exported by the server. The value 1 means only CORBA-compliant tags are exported. The value 0 (the default) means CORBA-compliant and non-compliant tags are exported.

Use value 1 when the server uses only SSL basic authentication for its security and clients (such as CICS or other OEM ORBs) use CORBA-compliant tags. This is only in the case when the server uses SSL basic authentication. If your server supports SSL client certificates as well, you do not have to set this variable.

Use value 0 (or take the default) when your server uses SSL basic authentication and interoperates with WebSphere clients on distributed platforms or WebSphere Application Server Enterprise Edition for OS/390 V3.02.

Example: `IBM_OMGSSL=1`

ICU_DATA=*path*
The path to binary files required by the XML Parser used by the System Management server during bootstrap and import server processing. If you installed the WebSphere for z/OS code in the default directory, you do not need to change this path. The default path is `/usr/lpp/WebSphere/bin/`.

Example: `ICU_DATA=/usr/lpp/WebSphere/bin/`

IIOP_SERVER_SESSION_KEEPALIVE=*n*
This variable, if set, defines the value in seconds provided to TCP/IP on the

SOCK_TCP_KEEPALIVE option for the IIOP listener. The function of this option is to verify if idle sessions are still valid by polling the client TCP/IP stack. If the client does not respond, the session is closed. If the client goes away without notifying the server, it would unnecessarily leave the session active on the server side. Use this option to clean up these unnecessary sessions. The default is zero.See TCP/IP APAR PQ18618 for more information about this option.

Notes:

1. If the environment variable is not set, the TCP/IP option is not set.
2. Setting the SOCK_TCP_KEEPALIVE option generates network traffic on idle sessions, which can be undesirable.

Example: `IIOP_SERVER_SESSION_KEEPALIVE=3600`

IR_GENERIC_SERVER_NAME=*SERVER_NAME*
The server name of the Interface Repository Server. The default is CBINTFRP. You must define a workload management (WLM) application environment using this name for the Interface Repository Server server regions to work.

IR_SPECIFIC_SERVER_NAME=*SERVER_INSTANCE_NAME*
A server instance name of the Interface Repository Server. The default is INTFRP01. You must specify this environment variable for all server instances in the second and subsequent systems in a sysplex.

IRPROC=*PROC_NAME*
The start procedure used by the Daemon Server to start the Interface Repository Server. The default is BBOIR. You can supply the name of your own start procedure. If you do so, copy the information from the default start procedure to your new start procedure.

Example: `IRPROC=BBOIR`

IVB_DEBUG_ENABLED=1
Enables the z/OS or OS/390 client and the application server to load the object level trace run time, and to use object level trace for tracing and/or debugging client and server application components. The value 1 is required for the application server, and for both C++ or Java clients running on z/OS or OS/390, when debugging C++ or Java business objects, servlets, JSPs, or Enterprise beans.

IVB_DRIVER_PATH=*path*
The name of the directory where WebSphere for z/OS files reside after SMP/E installation. The default is `/usr/lpp/WebSphere`.

Example: `IVB_DRIVER_PATH=/usr/lpp/WebSphere`

IVB_TRACE_HOST=*IP_ADDRESS (or HOSTNAME)*
Specifies the workstation IP address (or host name if you have the DNS server setup correctly) where the object level trace viewer runs. Use this when you are tracing and/or debugging your client and server components with the IBM Object Level Trace and Distributed Debugger Tools.

Example: `IVB_TRACE_HOST=MYHOST.IBM.COM`

IVB_TRACE_PORT=*port*
Specifies the same port as the TCP/IP port specified for the object level trace server. Use this when you are tracing and/or debugging your client and server components with the IBM Object Level Trace and Distributed Debugger Tools. The default is 2102.

Example: `IVB_TRACE_PORT=2102`

java.naming.security.credentials=*password*
The password used by the distinguished name specified by java.naming.security.principal. The password must match the password defined for the administrator access ID (default is WASAdmin) by the LDAP initialization file during initial system customization. IBM provides the WASAdmin access ID in a sample LDIF file called bboldif.cb. The default value is `secret`.

Example: `java.naming.security.credentials=secret`

Recommendation: You should change the IBM-supplied password.

java.naming.security.principal=*distinguished_name*
Distinguished name (user ID) defined to have write access to WsnName directory. Specify this only if you want to provide read/write access to all JNDI users. The distinguished name must match the one defined for the administrator access ID (default is WASAdmin) by the LDAP LDIF file during initial system customization. IBM provides the WASAdmin access ID in a sample LDAP initialization file called bboldif.cb. The default value is `cn=WASAdmin,o=WASNaming,c=us`.

Example:

```
java.naming.security.principal=cn=WASAdmin,o=WASNaming,c=us
```

Recommendation: We suggest you keep the WASAdmin access ID.

JAVA_COMPILER=
Specifies the use of the just-in-time (JIT) compiler.

If you use the environment variable, a null value (`JAVA_COMPILER=`) turns the JIT compiler on. Any other value turns the JIT compiler off.

By default, a Java virtual machine (JVM) running on z/OS or OS/390 uses the JIT compiler, so you do not have to explicitly set this environment variable. If you are debugging Java business objects or J2EE application components, however, turn off the JIT compiler by specifying a non-null value.

Example: `JAVA_COMPILER=NONE`

JAVA_IEEE754=EMULATION
Specifies the correct executable code for the system to load for the Java virtual machine (JVM) in which Java clients on z/OS or OS/390 run. This environment variable setting is required only for Java clients that run on z/OS or OS/390.

JVM_BOOTCLASSPATH=*path1***:***[path2]*
Enables the use of bootclasspath. This option is equivalent to the `-Xbootclasspath/p:` Java invocation option.

JVM_BOOTLIBRARYPATH=*path1***:***[path2]*
Enables the use of bootlibrarypath. This option is equivalent to the `-Dsun.boot.library.path=` Java invocation option.

| **JVM_DEBUG=1**
| This option is equivalent to the `–verbose:class,jni` Java invocation option. It
| reroutes JNI and class debug messages to SYSOUT for debugging purposes. Set
| JVM_DEBUG=1 to invoke JVM messaging.

| **Note:** Setting this variable does not result in garbage collection processing; to
| enable garbage collection, you must specify JVM_ENABLE_CLASS_GC=1.

JVM_DEBUG_PORT=*port*
: Specifies a TCP/IP port that the distributed debugger uses to connect to the JVM.

JVM_ENABLE_CLASS_GC=1
: Enables garbage collection of class objects when this environment variable is set to the value 1. Without this setting, garbage collection is not enabled for class objects, so the default behavior is equivalent to the `-Xnoclassgc` Java invocation option.

  If you need garbage-collection output in an output file, specify the filename through the IBM_JVM_ST_VERBOSEGC_LOG environment variable. Otherwise, garbage-collection output appears in SYSOUT for the server region.

JVM_ENABLE_VERBOSE_GC=1
: Sets verbose garbage collection on or off. The value 1 is required for enabling garbage collection messages. This option is equivalent to the `-verbose:gc` Java invocation option.

  If you need garbage-collection output in an output file, specify the filename through the IBM_JVM_ST_VERBOSEGC_LOG environment variable. Otherwise, garbage-collection output appears in SYSOUT for the server region.

JVM_EXTRA_OPTIONS=*string*
: Allows you to specify one new Java environment variable that is not already predefined by IBM (those predefined variables start with `JVM_`). With `JVM_EXTRA_OPTIONS`, *string* is the new Java option or property that you want to specify.

JVM_HEAPSIZE=*n*
: Sets the maximum size (in megabytes) of the JVM heap. The default is 256 MB. This option is equivalent to the `-Xmx=xxxM` Java invocation option.

  Example: `JVM_HEAPSIZE=256 # specifies a 256 MB heap`

JVM_LOCALREFS=
: Should only be used under the direction of IBM support. The default is 128.

JVM_LOGFILE=*filename*
: Specifies the HFS file in which JNI and class debug messages from the JVM will be logged.

  Recommendations:

  - Use this variable only in a single-server environment. If you use `JVM_LOGFILE` in a multiple-server environment, all the servers write to the same file, so you might have difficulty using the file for diagnostic purposes. In a multiple-server environment, use `JVM_DEBUG=1` to direct JNI and class debug messages to the SYSOUT for a specific server.
  - This log file does not contain garbage-collection output. If you enable garbage collection by specifying JVM_ENABLE_CLASS_GC=1, the output appears in SYSOUT for the server region, or in an HFS file you specify through the IBM_JVM_ST_VERBOSEGC_LOG environment variable. **Do not** specify the same HFS file for the `IBM_JVM_ST_VERBOSEGC_LOG` variable as you do for the `JVM_LOGFILE` variable. WebSphere for z/OS will not append data to an existing file; instead, the data will be overwritten if both of these variables specify the same HFS file.

JVM_MINHEAPSIZE=*n*
: Sets the mimimum size (in megabytes) of the JVM heap. The default is 256

MB. This option is equivalent to the `-Xms=xxxM` Java invocation option. For optimal performance, specify the same value for `JVM_HEAPSIZE` and `JVM_MINHEAPSIZE`.

LDAPBINDPW=*password*
: The password the Naming Server uses to bind to the LDAP server. Used in conjunction with LDAPNAME.

LDAPCONF=*filename*
: The LDAP configuration file used by WebSphere for z/OS. If you designate a file in the HFS, do not use quotes. If you designate an MVS data set, enclose the data set in single quotes.

Example: `LDAPCONF='bbo.s21slapd.conf'`

LDAPHOSTNAME=*name:port*
: The host name of the LDAP server that the Interface Repository Server uses as its data store.

LDAPIRBINDPW=*password*
: The password the Interface Repository Server uses to bind to the LDAP server. Used in conjunction with LDAPIRNAME.

LDAPIRCONF=*filename*
: The LDAP configuration file used by the LDAP server that the Interface Repository Server uses as its data store. If you designate a file in the HFS, do not use quotes. If you designate an MVS data set, enclose the data set in single quotes.

LDAPIRHOSTNAME=*name:port*
: The host name of the LDAP server that the Interface Repository Server uses as its data store.

LDAPIRNAME
: The LDAP entry name that the Interface Repository Server uses to authenticate itself to the LDAP server that it uses as its data store.

LDAPIRROOT=*root*
: The LDAP entry name at which the Interface Repository Server anchors its data.

Example: `LDAPIRROOT=o=BOSS,c=U`

LDAPNAME
: The LDAP entry name that the Naming Server uses to authenticate itself to the LDAP server that it uses as its data store.

LDAPROOT=*root*
: The LDAP entry name at which the Naming Server anchors its data.

Example: `LDAPROOT=o=BOSS,c=US`

LIBPATH=*path1***:***[path2]***:...**
: Specifies the DLL search paths for Java in the hierarchical file system (HFS). Specify system, WebSphere for z/OS, and Java DLLs.

Example:

```
LIBPATH=/db2_path/lib:/usr/lpp/java/J1.3/bin:/usr/lpp/java/J1.3/bin/classic:/usr/lpp/WebSphere/lib
```

where *db2_path* is the HFS where you installed DB2.

LOGSTREAMNAME=*LOG_STREAM_NAME*
: The WebSphere for z/OS error log stream name the Daemon and System Management servers use during bootstrap. If not specified in the environment

file for the Daemon and System Management servers during bootstrap, the system uses the following algorithm to form an error log stream name. WebSphere for z/OS:

1. Takes the first qualifier in the Daemon Server’s IP name.
2. If the first qualifier is more than 8 characters, divides the qualifier into 8-character strings and separates them with periods.
3. Adds a high-level qualifier “BBO”.

For example, if the Daemon IP name is MYDAEMONSERVER.IBM.COM, the algorithm would produce an error log stream name BBO.MYDAEMON.SERVER.

After bootstrap, you can create or change an error log stream name for the entire sysplex, a server, or a server instance through the Administration application. A server error log stream setting overrides the general WebSphere for z/OS setting, and a server instance setting overrides a server setting. Thus, you can set up general error logging, but direct error logging for servers or server instances to specific log streams.

During processing, if the specified log stream is not found or not accessible, a message is issued and errors are written to the server’s joblog.

Example: `LOGSTREAMNAME=MY.CB.ERROR.LOG`

Tip: Do not put the log stream name in quotes. Log stream names are not data set names.

MAX_SRS=*nn*

Specifies the total number of server regions allowed by workload management to run concurrently in the server’s application environment. That is, workload management will not start more server regions for a particular application environment than are specified through this environment variable.

Use this environment variable to limit the number of server regions created by workload management for a server. The default is zero, which means there is no limit.

Attention: If you specify MAX_SRS, you must ensure that you specify a MAX_SRS value that is greater than or equal to MIN_SRS times the number of service classes you have defined for this application environment. Failure to do so can result in timeouts due to an insufficient number of server regions.

Example: `MAX_SRS=10`

MIN_SRS=*nn*

The number of server regions to be kept running once those server regions have initialized. That is, workload management will not direct the server region to shut down even though it becomes inactive. Use this environment variable when the response time for the workload requires that several server regions are always ready to process work.

The default for J2EE servers is 1. For MOFW servers, the default is 0. The maximum value is 20. If you specify more than 20, the variable is set to 20.

WebSphere for z/OS garbage collection may cause a server region to refresh, but the minimum number of server regions will not fall below the value specified on this environment variable.

Example: `MIN_SRS=2`

NM_GENERIC_SERVER_NAME=*SERVER_NAME*
The server name of the Naming Server. The default is CBNAMING. You must define a workload management (WLM) application environment using this name for the Naming Server server regions to work.

Example: `NM_GENERIC_SERVER_NAME=CBNAMING`

NM_SPECIFIC_SERVER_NAME=*SERVER_INSTANCE_NAME*
The server instance name of the Naming Server. The default is NAMING01. You must specify this environment variable for all server instances in the second and subsequent systems in a sysplex.

Example: `NM_SPECIFIC_SERVER_NAME=NAMING01`

NMPROC=*PROC_NAME*
The start procedure used by the Daemon Server to start the Naming Server. The default is BBONM. You can supply the name of your own start procedure. If you do so, copy the information from the default start procedure to your new start procedure.

Example: `NMPROC=BBONM`

OTS_DEFAULT_TIMEOUT=*n*
The amount of time (in seconds) given by default to an application transaction to complete. This amount of time is given to the application transaction if it does not set its own time-out value through the `current –> set_timeout` method.

The default is 30 seconds and the maximum value is 2147483 seconds (24.85 days). You should not use a null or 0 value.

Note: When a conversation is activated, the system performs special processing for the System Management server instances **only**.
- If the OTS_DEFAULT_TIMEOUT variable is not set, it is added.
- If the value for OTS_DEFAULT_TIMEOUT is less than 3600 (seconds), it is set to 3600.

This special processing is performed for the System Management server instances because the server instances sometimes perform long-running transactions. Other server instances do not require such lengthy transaction defaults.

Example: `OTS_DEFAULT_TIMEOUT=30`

OTS_MAXIMUM_TIMEOUT=*n*
The maximum allowable amount of time (in seconds) given to an application transaction to complete. If an application assigns a greater amount of time, the system limits the time to the OTS_MAXIMUM_TIMEOUT value.

The default is 60 seconds and the maximum value is 2147483 seconds (24.85 days). You should not use a null or 0 value.

Note: When a conversation is activated, the system performs special processing for the System Management server instances **only**.
- If the OTS_MAXIMUM_TIMEOUT variable is not set, it is added.
- If the value for OTS_MAXIMUM_TIMEOUT is less than 3600 (seconds), it is set to 3600.

This special processing is performed for the System Management server instances because the server instances sometimes perform long-running transactions. Other server instances do not require such lengthy transaction defaults.

Example: `OTS_MAXIMUM_TIMEOUT=60`

PATH=*path*
Specifies the path.

RAS_MINORCODEDEFAULT=*value*
Determines the default behavior for gathering documentation about system exception minor codes. Use only under the guidance of IBM Service.

CEEDUMP
Captures callback and offsets.

Tip: It takes time for the system to take CEEDUMPs and this may cause transaction timeouts. For instance, your OTS_DEFAULT_TIMEOUT may be set to 30 seconds, but, since taking a CEEDUMP can take longer than 30 seconds, your application transaction may time out. To prevent this from happening, either:

- Increase the transaction timeout value.

  or
- Code RAS_MINORCODEDEFAULT=NODIAGNOSTICDATA. Be sure TRACEMINORCODE is **not** in the environment file.

TRACEBACK
Captures Language Environment and z/OS UNIX traceback data.

SVCDUMP
Captures an MVS dump (but will not produce a dump in the client).

NODIAGNOSTICDATA
The default. This setting will not cause the gathering of a CEEDUMP, TRACEBACK, or SVCDUMP.

Note: Sometimes results depend on the setting of another environment variable, TRACEMINORCODE. If you code TRACEMINORCODE=(null value) and RAS_MINORCODEDEFAULT=TRACEBACK you get a traceback. But, if you code RAS_MINORCODEDEFAULT=NODIAGNOSTICDATA and TRACEMINORCODE=ALL, you also get a traceback. So, specifying RAS_MINORCODEDEFAULT=NODIAGNOSTICDATA does not cancel TRACEBACK; it simply does not cause a TRACEBACK to be gathered.

RECOVERY_TIMEOUT=*n*
The time in minutes that this control region uses to attempt to resolve transactions before asking the installation if it should:

1. Give up trying to resolve,
2. Write transaction-related information to the joblog or hard copy log, then
3. Terminate.

If the installation replies that it would like the recovery to continue, the control region will attempt recovery for another *n* minutes before re-issuing the WTOR. Of course, once all the transactions are resolved, the control region will just terminate and you won't see the WTOR again. This variable applies only to control regions in restart and recovery mode, when you are not running on your configured system. The default value is 15 minutes.

Example:

```
RECOVERY_TIMEOUT=7
```

RECYCLE_J2EE_SERVERS=Y|N
Specifies whether or not the Systems Management server will automatically recycle a J2EE server instance when a modified conversation for the associated J2EE server is activated. If **Y** is specified, the Systems Management server will automatically recycle J2EE server instances whenever a modified conversation is activated. When **Y** is specified, the environment variable performs the same function that was implemented prior to the introduction of this new variable. If **N** is specified, the Systems Management server:

- Will not automatically recycle J2EE server instances whenever a modified conversation for the associated J2EE server is activated. The servers must still be restarted (manually or with some customer-provided automation) to allow changes to the server instance to take effect.
- Will not delete files associated with deleted applications on these server instances from the HFS because the application will still be running until the server instance is restarted. To prevent accumulation of obsolete files on the HFS, you should delete all files and directories associated with the deleted applications after restarting the server instances. This includes:

```
 * /<CBCONFIG>/apps/<server_name>/A/A<uuid>/  (directory)
* /<CBCONFIG>/apps/<server_name>/A/A<uuid>.*  (files)
* /<CBCONFIG>/apps/<server_name>/<j2ee_application_name>/   (directory)
* /<CBCONFIG>/working/<server_name>/temp/<sysplex_name>/
          <server_inst_name>/<app_name>/   (directory
```

Note: RECYCLE_J2EE_SERVERS is only used by the Systems Management server. If specified on any other server, it is ignored.

Default: Y

Example: RECYCLE_J2EE_SERVERS=Y

REM_DCEPASSWORD=*password*
The password of the remote DCE principal passed in the security context when an z/OS or OS/390 client makes a request to a system outside the sysplex and SSL Type 1 authentication is being used. The password must conform to DCE requirements for passwords.

Example: `REM_DCEPASSWORD=mydcePW`

REM_DCEPRINCIPAL=*principal*
The principal passed in the security context when a client makes a request to a system outside the sysplex and SSL Type 1 authentication is being used. This principal must be defined on the target server. The value must conform to DCE requirements for principals.

Example: `REM_DCEPRINCIPAL=myDCEprin`

REM_PASSWORD=*password*
The password used in the security context when a client makes a request to a remote z/OS or OS/390 system and user ID/password security or SSL security is being used.

Example: `REM_PASSWORD=MYPASSW`

REM_USERID=*USER_ID*
The user ID used in the security context when a client makes a request to a remote z/OS or OS/390 system and user ID/password security or SSL security is being used.

Example: `REM_USERID=MCOX`

RESOLVE_IPNAME=*IP_NAME*
The Internet Protocol name that the System Management Server registers with the Domain Name Service (DNS). Any CORBA client communication with WebSphere for z/OS requires this IP Name. If not set, the Resolve IP Name is the system on which the program is running.

Rule: The value for RESOLVE_IPNAME should be a fully-qualified name, but it cannot exceed 255 characters.

Example: `RESOLVE_IPNAME=CBQ091.COMPANY.NY.COM`

RESOLVE_PORT=*n*
The port number at which the System Management Server listens for requests. The default is 900. This is a well-known port for Object Request Brokers, so IBM advises that you do not change this variable. If you already have an application that uses this port, consider using TCP/IP bind-specific support and the SRVIPADDR environment variable.

Example: `RESOLVE_PORT=900`

| **SESSION_COOKIE_NAME=**
| Specifies the name of the cookie that is to be used for this J2EE server instance,
| if cookies are enabled. The cookie name must only contain:

- English alphanumeric characters (uppercase or lowercase A to Z and numbers 0 to 9)
- Underscore (_)
- Period (.)
- Hyphen (-)

| The default value is "JSESSIONID".

| **Note:** The value specified on this environment variable must match the value specified on the `session.cookie.name` property in the webcontainer.conf file.

SM_DEFAULT_ADMIN=*USER_ID*
The user ID for the administrator who uses the Administration and Operations applications. This environment variable is used by the System Management bootstrap during installation—setting this environment variable after the System Management bootstrap runs has no effect. If you do not define this environment variable, the default user ID is CBADMIN. You must define this user ID to z/OS or OS/390 and give it appropriate security authorizations (for example, RACF permissions and LDAP permissions).

Note: After the System Management bootstrap runs, you can define additional administrator user IDs only through the Administration application. Those user IDs do not replace the user ID defined by SM_DEFAULT_ADMIN.

Example: `SM_DEFAULT_ADMIN=DUDE`

SM_GENERIC_SERVER_NAME=*SERVER_NAME*
The server name of the Systems Management Server. The default is CBSYSMGT. You must define a workload management (WLM) application environment using this name for the Systems Management Server server regions to work.

Example: `SM_GENERIC_SERVER_NAME=CBSYSMGT`

SM_SPECIFIC_SERVER_NAME=*SERVER_INSTANCE_NAME*
The server instance name of the Systems Management Server. The default is SYSMGT01. You must specify this environment variable for all server instances in the second and subsequent systems in a sysplex.

Example: `SM_SPECIFIC_SERVER_NAME=SYSMGT01`

SMPROC=*PROC_NAME*
The start procedure used by the Daemon Server to start the Systems Management Server. The default is BBOSMS. You can supply the name of your own start procedure. If you do so, copy the information from the default start procedure to your new start procedure.

Example: `SMPROC=BBOSMS`

SOMOOSQL=*value*
Improves performance for client applications that use object-oriented SQL queries on string attributes. By using SOMOOSQL=1, string comparisons are pushed down to the database.

The default value is null (SOMOOSQL=).

Rule: You can use SOMOOSQL=1 only when the database and server region address spaces have been declared to run in the same locale.

SRVIPADDR=*IP_ADDRESS*
The IP address in dotted decimal format that WebSphere for z/OS servers use to listen for client connection requests.

This IP address is used by the server to bind to TCP/IP. Normally, the server will listen on all IP addresses configured to the local TCP/IP stack. However if you want to fence the work or allow multiple heterogeneous servers to listen on the same port, you can use SRVIPADDR. The specified IP address becomes the only IP address over which WebSphere for z/OS receives inbound requests. Normally, you also have to map the Daemon IP name, resolve IP name, or host name of the server that you are on to this particular SRVIPADDR.

| **SSL_HANDSHAKE_THREAD_COUNT=***n*
| Specifies the number of SSL handshake threads that are present in the control
| region. The default is 3.

| **Example:** `SSL_HANDSHAKE_THREAD_COUNT=10`

SSL_KEYRING=*keyring*
The name of the z/OS or OS/390 client's key ring used in SSL processing. This key ring must reside in RACF.

Example: `SSL_KEYRING=IVPRING`

| **SSL_SERVER_V3CIPHERS=***string*
| Defines the SSL Version 3 cipher suites that system SSL uses in the SSL
| handshake for an SSL connection. It overrides any server-wide setting set via
| the Administration Application. Specify a string as documented in ″z/OS
| System Secure Sockets Layer Programming″ (SC24-5901). Each cipher is
| represented by two characters (for example, ″09″ instead of ″9″). You can
| specify the string with or without comma delineation. If you delineate with
| commas, a validity check will run against the installed ciphers. The default is
| an empty string, meaning no change is made to the cipher suites.

| **Examples:**

```
SSL_SERVER_V3CIPHERS=09,0A,05
SSL_SERVER_V3CIPHERS=090A05
```

SYS_DB2_SUB_SYSTEM_NAME=*NAME*
: The DB2 name used by Daemon and System Management servers to connect to the database. Use either the DB2 subsystem name or group attachment name. The default is DB2. If the default is not correct for your installation, change the environment variable to match the correct value.

  Example: `SYS_DB2_SUB_SYSTEM_NAME=DB21`

TRACEALL=*n*
: Specifies the default tracing level for WebSphere for z/OS. Valid values and their meanings are:

0	No tracing
1	Exception tracing, the default
2	Basic and exception tracing
3	Detailed tracing, including basic and exception tracing

  Use this variable in conjunction with the TRACEBASIC and TRACEDETAIL environment variables to set tracing levels for WebSphere for z/OS subcomponents. Do not change this variable unless directed by IBM service personnel.

  Example: `TRACEALL=1`

TRACEBASIC=*n* | (*n*,...)
: Specifies tracing overrides for particular WebSphere for z/OS subcomponents. Subcomponents, specified by numbers, receive basic and exception traces. If you specify more than one subcomponent, use parentheses and separate the numbers with commas. Contact IBM service for the subcomponent numbers and their meanings. Other parts of WebSphere for z/OS receive tracing as specified on the TRACEALL environment variable. Do not change TRACEBASIC unless directed by IBM service personnel.

  Example: `TRACEBASIC=3`

TRACEBUFFCOUNT=*n*
: Specifies the number of trace buffers to allocate. Valid values are 4 through 8. The default is 4.

TRACEBUFFLOC=SYSPRINT | BUFFER
: Specifies where you want trace records to go: either to sysprint (SYSPRINT) or to a memory buffer (BUFFER), then to a CTRACE data set. The default is to direct trace records to sysprint for the client and to a buffer for all other WebSphere for z/OS processes. For servers, you may specify one or both values, separated by a space. For clients, you may specify TRACEBUFFLOC=SYSPRINT only.

  Example: `TRACEBUFFLOC=SYSPRINT BUFFER`

TRACEBUFFSIZE=*n*
: Specifies the size of a single trace buffer in bytes. You can use the letters “K” (for kilobytes) or “M” (for megabytes). Valid values are 128K through 4M. The default is 1M.

TRACEDETAIL=*n* | (*n*,...)
: Specifies tracing overrides for particular WebSphere for z/OS subcomponents. Subcomponents, specified by numbers, receive detailed traces. If you specify more than one subcomponent, use parentheses and separate the numbers with commas. Contact IBM service for the subcomponent numbers and their

meanings. Other parts of WebSphere for z/OS receive tracing as specified on the TRACEALL environment variable. Do not change TRACEDETAIL unless directed by IBM service personnel.

Examples:

```
TRACEDETAIL=3
```

```
TRACEDETAIL=(3,4)
```

TRACEMINORCODE=*value*
Enables traceback of system exception minor codes. Use only when instructed by IBM Service. Values are:

ALL|all
Enables traceback for all system exception minor codes.

minor_code
Enables traceback for a specific minor code. Specify the code in hex, such as X'C9C21234'.

(null value)
The default. This setting will not cause gathering of a traceback.

Note: Sometimes results depend on the setting of another environment variable, RAS_MINORCODEDEFAULT. If you code TRACEMINORCODE=ALL and RAS_MINORCODEDEFAULT=NODIAGNOSTICDATA, you get a traceback. But, if you code TRACEMINORCODE=(null value) and RAS_MINORCODEFAULT=TRACEBACK you also get a traceback. So, specifying TRACEMINORCODE=(null value) does not cancel TRACEBACK; it simply does not cause a TRACEBACK to be gathered.

TRACEPARM=*SUFFIX* | *MEMBER_NAME*
Identifies the CTRACE PARMLIB member. The value can be either a two-character suffix, which is added to the string CTIBBO to form the name of the PARMLIB member, or the fully-specified name of the PARMLIB member. For example, you could use the suffix "01", which the system resolves to "CTIBBO01". A fully-specified name must conform to the naming requirements for a CTRACE PARMLIB member. For details, see *z/OS MVS Diagnosis: Tools and Service Aids*, GA22-7589.

The default value is 00.

If this environment variable is specified and the PARMLIB member is not found, the default PARMLIB member, CTIBBO00, is used. If neither the specified nor the default PARMLIB member is found, tracing is defined to CTRACE, but there is no connection to a CTRACE external writer. For details on the PARMLIB member and the use of the CTRACE external writer, see *WebSphere Application Server V4.0.1 for z/OS and OS/390: Messages and Diagnosis*, GA22-7837.

Note that the Daemon Server is the only server that recognizes this environment variable.

Example: `TRACEPARM=01`

TRACESPECIFIC=*n* | **(***n***,...)**
Specifies tracing overrides for specific WebSphere for z/OS trace points. Trace points are specified by 8-digit, hexadecimal numbers. To specify more than one trace point, use parentheses and separate the numbers with commas. You can also specify an environment variable name by enclosing the name in single

quotes. The value of the environment variable will be handled as if you had specified that value on TRACESPECIFIC. Do not use TRACESPECIFIC unless directed by IBM service personnel.

Examples:

```
TRACESPECIFIC=03004020

TRACESPECIFIC=(03004020,04005010)

TRACESPECIFIC='xyz' [where xyz is an environment variable name]

TRACESPECIFIC=('xyz','abc',03004021)
[where xyz and abc are environment variable names]
```

WAS_JAVA_OPTIONS=-*option1* **-***option2* **-***option3*
Should be used only under the direction of IBM support. The default is null.

WS_EXT_DIRS=*name:name: ...*
Specifies the common JAR files and directories for extensions to the run-time functions or configuration of a J2EE server instance. For example, if you are configuring Type 4 JDBC connector for Enterprise bean or servlet use, you use `WS_EXT_DIRS` to specify the location of the connector's JDBC resource factory jar file. Each JAR file and directory in the list is separated with a colon (:). These files are loaded into the WebSphere for z/OS run-time by the Web Container run-time class loader.

Note: See the related information about WebSphere for z/OS class loaders and application modules in "Overview of WebSphere for z/OS classloader operation" on page 120.

Example: `WS_EXT_DIRS=/tmp/OracleJdbcResourceFactory.java`

JVM properties and properties files

Use a properties file only if you want to change the default settings that WebSphere for z/OS uses for the Java virtual machine (JVM) that runs in the server. Note that the property settings that you define in this file override any environment variables that you set through the WebSphere for z/OS Administration application.

JVM properties are similar to environment variables and have a similar syntax. However, upper and lower case characters are significant.

The syntax of the JVM properties has this pattern:

```
property=value
```

Where:

property
is the JVM property.

value
is the setting for the property. The descriptions define possible values for each property.

Note: Do **not** place quotes arround the values.

How to manage JVM properties

To change default properties for the JVM in a particular server, create a file in the same HFS directory in which WebSphere for z/OS places the current.env file containing environment variable settings for the server:

```
CBCONFIG/controlinfo/envfile/SYSPLEX/SRVNAME/
```

where

&CBCONFIG
: Is the read/write directory that you specify at installation time as the directory into which WebSphere for z/OS is to write configuration data and environment files.

&SYSPLEX
: Is the name of your sysplex.

&SRVNAME
: Is the server instance name.

Note: This subdirectory will not exist until the conversation containing this server is activated for the first time.

Rules:
- The file must be named `jvm.properties`
- The permission bits for this HFS directory should be 775 so that server region user IDs have read access to the directory.

You may add your own, installation-defined JVM properties to this file, as long as you follow the syntax convention and rules. You cannot alter these properties using the Administration application; instead, you must hand-edit the `jvm.properties` file.

To retrieve the values of these properties, you can use one of the following:
- The `System.getProperty` method, through which any applications running in the server can obtain the values of properties specified in this file.
- Deployment descriptors that can be retrieved through ctx.lookup("java:comp/env/*my_app_variable*")

  where *my_app_variable* is a specific value that you can assign during application deployment.

  This approach to retrieving system properties avoids possible conflicts among multiple applications running in the same server.

JVM property use

Table 26 lists the supported JVM properties for a WebSphere for z/OS server. The following list explains the table contents:
- "O" means optional
- A blank in the Default column means the variable is not set
- A blank in other columns means the variable is not used.

Table 26. Where to use JVM properties

JVM property=<default>	J2EE server instance	MOFW server instance
com.ibm.CORBA.iiop.noLocalCopies=	O	

Table 26. Where to use JVM properties (continued)

JVM property=<default>	J2EE server instance	MOFW server instance
com.ibm.ejs.EJBCache.size=251	O	
com.ibm.websphere.cmp.cache.maxLevels=1	O	
com.ibm.websphere.cmp.cache.printlevels=0	O	
com.ibm.websphere.cmp.connection.policy=acrossTrans	O	
com.ibm.websphere.cmp*Location Name*.connection.timeout=0	O	
com.ibm.websphere.cmp.*Location Name*.preparedStatement.poolsize=100	O	
com.ibm.websphere.cmp.*Location Name*.connection.maxTrans=100	O	
com.ibm.websphere.persistence.bean.managed.custom.finder.access.intent=0	O	
com.ibm.websphere.preconfiguredCustomServices=	O	
com.ibm.websphere.security.AuthorizationTable	O	
com.ibm.websphere.sendredirect.compliance=false	O	
com.ibm.ws.classloader.ejbDelegationMode=true	O	
com.ibm.ws.classloader.J2EEApplicationMode=false	O	
com.ibm.ws.classloader.warDelegationMode= false	O	
com.ibm.ws390.ConnectionUsageScopeDefault=	O	
com.ibm.ws390.install.root=/usr/lpp/WebSphere	O	
com.ibm.ws390.server.classloadermode=2	O	
com.ibm.ws390.trace.settings=	O	O
com.ibm.ws390.wc.config.dynxmlfilename=*path*.dynacache.xml	O	
com.ibm.ws390.wc.config.dynsrvxmlfilename=*path*/servletcache.xml	O	
com.ibm.ws390.wc.config.filename=	O	
com.sun.tools.javac.main.largebranch=	O	
invocationCacheSize=	O	
mail.debug=	O	
org.omg.CORBA.ORBInitialHost=RESOLVE_IPNAME	O	O
org.omg.CORBA.ORBInitialPort=900	O	O
service.debug.enabled=false	O	
WEB_SECURITY_VERSION=	O	

Properties descriptions

com.ibm.CORBA.iiop.noLocalCopies=*string*

Determines whether objects passed between enterprise beans running in the same JVM are passed by reference instead of by value.

By default, objects are passed by value. If you specify any non-null value for the variable *string*, objects will be passed by reference.

Recommendation: Use "true" as the assignment, which is consistent with WebSphere on distributed platforms.

Attention: Passing objects by reference allows the caller to observe changes made to parameters by the called method. Though passing objects by reference results in a significant performance improvement, it is at the cost of removing data integrity safeguards. Use this option with extreme caution.

Example:

```
com.ibm.CORBA.iiop.noLocalCopies=true
```

com.ibm.ejs.EJBCache.size=*number*
Sets the size of the Enterprise bean (EJB) cache.

Recommendation: Calculate the cache size based on the workload of this J2EE server. The number that you specify for the size should correspond to the number of activated entity beans in the J2EE server.

Default: 251

com.ibm.websphere.cmp.cache.maxLevels=*number*
Specifies the maximum number of levels of stacked transactions for which the J2EE server may cache container-managed persistence (CMP) connections and prepared statements. Specifying a value of 0 causes the J2EE server to cache CMP connections and prepared statements for only the root transaction. For additional information, see the performance tuning topics in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Operations and Administration*, SA22-7835.

Default: 1 (caching occurs for the root transaction plus one level of stacked transactions)

com.ibm.websphere.cmp.cache.printlevels=*number*
Specifies the level for which the J2EE server is to print debugging information related to caching for container-managed persistence (CMP) connections and prepared statements. If you specify 3, for example, the J2EE server will print debugging information each time an application exceeds three levels of stacked transactions beyond the root transaction.

Default: 0 (no debugging information is printed)

com.ibm.websphere.cmp.connection.policy=*string*
Sets the WebSphere container managed persistence (CMP) connection and prepared statement pooling behavior of a transaction. Possible values, which represent the two pooling models, are *acrossTrans* (the default) and *inTrans*. Change this property when you want to manage the performance of an application that uses prepare statements.

acrossTrans
The *acrossTrans* pooling model employs a thread-based pool for pooling connections and prepared statements. Any request for connections and prepared statements by the EJB Container on behalf of CMP entity beans accessed within a given transaction is satisfied from the current thread's connection and statement pool. At the end of a transaction, all connections and prepared statements are returned to the pool. Only one connection per DB2 subsystem is pooled per thread. This is the default pooling model and generally provides the best performance.

inTrans
The *inTrans* pooling model employs a transactional pool for pooling connections and prepared statements. Any requests for connections and prepared statements by the EJB Container on behalf of CMP entity beans accessed within a given transaction are satisfied from that transaction's connection and statement pool. At the end of a transaction, all connections

and prepared statements in the pool are closed. This pooling model provides the most frugal management of database resources.

Example:

```
com.ibm.websphere.cmp.connection.policy=acrossTrans
```

com.ibm.websphere.cmp.*Location-Name***.connection.timeout=***time*

Specifies the timeout value for pooled connections. If a connection is idle longer than the specified time, it will be closed and removed from the pool. This detection occurs only at the time the EJB container attempts to use a pooled connection. If the pool detects the only available connection has timed out, it is closed and a new connection is created. The variable text, *Location-Name*, correlates a property setting to a specific datasource definition from the Administration application. It matches the ″Location Name″ value specified on the datasource definition from the Administration application. Default for this property is 0, which means no timeout.

Note: This property setting is useful only for connections that are used to access another DB2 subsystem (i.e. DRDA). Connection to another DB2 subsystem can timeout as specified by DB2's idle connection timeout setting.

Recommendation: If your DB2 idle connection timeout setting is greater than 0, IBM recommends that you set your WebSphere CMP connection timeout value to 90% of your DB2 idle connection timeout value.

Example:

```
com.ibm.websphere.cmp.Location-Name.connection.timeout=0
```

com.ibm.websphere.cmp.*Location-Name***.preparedStatement.poolsize=***size*

Sets the prepared statement pool size. The variable text, *Location-Name*, correlates a property setting to a specific datasource definition from the Administration application. It matches the ″Location Name″ value specified on the datasource definition from the Administration application. Generally, there is no need to modify this setting. Default for this property is 100.

Attention: Prepared statements consume DB2 sections and cursors. The DB2 default number of sections is 150, and the default number of cursors per connection is 100. The prepared statement poolsize value must not exceed the lesser of these two numbers. The settings for these numbers are contained in the JDBC profile, which is created by the db2genJDBC utility.

Example:

```
com.ibm.websphere.cmp.Location-Name.preparedStatement.poolsize=100
```

com.ibm.websphere.cmp.*Location-Name***.connection.maxTrans=***#ofTransactions*

Specifies the number of transactions in which a connection may participate before it is closed. DB2 collects SMF statistics for connections only at the time they are closed. Use this setting to control how many transactions can be executed to collect SMF statistics through a connection before it is closed. The variable text, *Location-Name*, correlates a property setting to a specific datasource definition from the Administration application. It matches the ″Location Name″ value specified on the datasource definition from the Administration application. Default for this property is 100.

Example:

```
com.ibm.websphere.cmp.Location-Name.connection.maxTrans=100
```

com.ibm.websphere.persistence.bean.managed.custom.finder.access.intent=*value*
Specifies the access intent of custom-finder methods for a bean that uses container-managed persistence (CMP bean). The values you may specify are 0 (update access intent) or 1 (read-only access intent). Using this property in the J2EE server's JVM property file sets the access intent for all CMP beans that run in the server.

Rule: Some applications use custom finders that contain the `FOR UPDATE` clause, the `ORDER BY` keyword, or `DISTINCT` keyword on the `SELECT` operation. In such cases, specify the value 1 to avoid encountering an SQL error (`SQLCODE` -126) when the J2EE server attempts to run the CMP bean. If you do not want this setting used for all CMP beans that run in the J2EE server, see the "Verify access intent" item in "Checklist for using pessimistic concurrency control" on page 58 for alternative choices.

Note: Using a setting of 1 (read-only access intent) does not prevent the custom finder from making updates, should any be necessary during the course of its processing.

Default: 0 (update access intent)

com.ibm.websphere.preconfiguredCustomServices=*pathname*

Specifies customer-provided services to be installed in the Java virtual machine that runs under the J2EE server. The value you specify for this property is the full directory path and name of an ASCII xml file for the custom service. You may specify more than one xml file, provided you follow each xml file name with a colon. Each xml file may contain more than one custom service application.

For information about custom services and their xml files, see the topic about developing custom services in the InfoCenter for WebSphere Application Server Advanced Edition Version 4.0. The InfoCenter is available at

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

Examples:

```
com.ibm.websphere.preconfiguredCustomServices=/usr/lpp/services/service1.xml
com.ibm.websphere.preconfiguredCustomServices=/services/service1.xml:/services/service2.xml:
```

com.ibm.websphere.security.AuthorizationTable=<implementation_classs>
Specifies the implementation class for the authorization table. The default implementation class provided with WebSphere for z/OS is:

`com.ibm.ws390.wc.security.AuthorizationTableImpl`

Example:

```
com.ibm.websphere.security.AuthorizationTable=
     com.ibm.ws390.wc.security.AuthorizationTableImpl
```

com.ibm.websphere.sendredirect.compliance=
Specifies whether or not a URI is appended to the WebSphere for z/OS install_root or the Web application's context root when the URI starts with a slash. If the URI starts with a slash, and the sendRedirect property is set to true, the URI is appended to the install_root; if the URI starts with a slash, and the sendRedirect property is set to false, the URI is appended to the Web application's context root, The default value is false.

Example: `com.ibm.websphere.sendredirect.compliance=false`

com.ibm.ws.classloader.ejbDelegationMode=*true*|*false*
Specifies the behavior of particular WebSphere for z/OS class loaders during the search for a class in a JAR module. This JVM property setting determines whether a class loader first searches its own classpath before delegating to its parent, or immediately delegates to its parent class loader. With the default

value, the class loader immediately delegates to its parent class loader during the search for an EJB class. For additional information about classloader relationships and operation, see "Overview of WebSphere for z/OS classloader operation" on page 120.

Default: true (the class loader immediately delegates to its parent class loader)

Example: `com.ibm.ws.classloader.ejbDelegationMode=false`

com.ibm.ws.classloader.J2EEApplicationMode=*true***|***false*
Specifies the type and behavior of class loaders that the J2EE server uses to load application modules, in accordance with the Sun Microsystems J2EE 1.3 specification. For additional information about application packaging and classloader operation, see "Overview of WebSphere for z/OS classloader operation" on page 120.

Default: false

Example: `com.ibm.ws.classloader.J2EEApplicationMode=true`

com.ibm.ws.classloader.warDelegationMode=*true***|***false*
Specifies the behavior of particular WebSphere for z/OS class loaders during the search for a class in a WAR module. This JVM property setting determines whether a class loader first searches its own classpath before delegating to its parent, or immediately delegates to its parent class loader. For additional information about classloader relationships and operation, see "Overview of WebSphere for z/OS classloader operation" on page 120.

Default: true only if `com.ibm.ws.classloader.J2EEApplicationMode` is set to true; otherwise, the default is false. If the value is true, the class loader immediately delegates to its parent class loader; if the value is false, the class loader searches its own classpath before delegating to its parent.

Example: `com.ibm.ws.classloader.warDelegationMode=true`

com.ibm.ws390.ConnectionUsageScopeDefault=
Enables the reuse of physical connections to J2EE connectors associated with this J2EE server. Setting this JVM property to the value `SeriallyReusable` causes the J2EE server to create a connection pool that is associated with a single global transaction, and to return connections to the pool when the J2EE application component closes the connection. If this property is set to another value, or is not set at all, the J2EE server cannot reuse a connection until the global transaction is either committed or rolled back.

For further information about using this JVM property, see "Connection pooling and reuse" on page 72.

Example: `com.ibm.ws390.ConnectionUsageScopeDefault=SeriallyReusable`

com.ibm.ws390.install.root=
Specifies the fully qualified directory path and file name for the HFS on which WebSphere for z/OS is installed.

Example: `com.ibm.ws390.install.root=/usr/lpp/WebSphere`

com.ibm.ws390.server.classloadermode=*number*
Specifies the type and behavior of class loaders that the J2EE server uses to load a class from an application module. This capability supports different approaches to packaging application components for installation in a J2EE server, and influences the search-path order that WebSphere for z/OS class loaders use to find and load classes. For additional information about application packaging guidelines and classloader operation, see "Overview of WebSphere for z/OS classloader operation" on page 120.

Recommendation: For most applications, use the default value (2, application mode).

Valid values for this property are:

0 Specifies module mode.

Recommendation: Use this value only if your applications have Manifest classpath statements in EJB JAR files or in WAR files. Even in this case, IBM recommends that you change this property setting to 2 (application mode).

With module mode, WebSphere for z/OS uses only one classloader per module. Each module (JAR or WAR file) has its own unique classloader. Visibility of other modules in the application is achieved only when Manifest classpath entries are added to a module.

If you specify module mode, you must include all application dependent files in each application's EAR file, or place them in the directory specified through the `APP_EXT_DIR` environment variable. Any JAR or zip files that exist in the directory specified on this environment variable are considered common files and can be accessed by any application running in the same J2EE server.

1 Specifies compatibility mode, which allows compatibility with applications from previous releases of WebSphere for z/OS. In this mode, all EJB module classloaders have visibility of all other EJB module classloaders, and all Web application modules have visibility of the EJB classloaders. The EJB classloaders are searched in the order in which the EJB modules were initialized.

If you specify compatibility mode, you must include all application dependent files in each application's EAR file, or place them in the directory specified through the `APP_EXT_DIR` environment variable. Any JAR or zip files that exist in the directory specified on this environment variable are considered common files and can be accessed by any application running in the same J2EE server.

2 Specifies application mode, which allows all classloaders in a J2EE application to have visibility of other classloaders in the same application. The search order is the same as the order in which the modules are defined in the application.xml for the EAR file.

If you specify application mode, you must include all application dependent files in each application's EAR file, or place them in the directory specified through the `APP_EXT_DIR` environment variable. Any JAR or zip files that exist in the directory specified on this environment variable are considered common files and can be accessed by any application running in the same J2EE server.

3 Specifies server mode, which allows all application classloaders on a J2EE server to have visibility to all other application classloaders in the server. This setting enables classes in one application to be visible to classes in all of the other applications residing on that server. When server mode is specified, common files do not have to be placed in the directory specified through the `APP_EXT_DIR` environment variable.

Default: 2 (application mode)

Example: `com.ibm.ws390.server.classloadermode=1`

com.ibm.ws390.wc.config.dynxmlfilename=*path*.dynacache.xml
Specifies the location of your dynacache.xml file.

Example:
`com.ibm.ws390.wc.config.dynxmlfilename=/u/SVR1/BBOASR4/dynacache.xml`

com.ibm.ws390.wc.config.dynsrvxmlfilename=*path*/servletcache.xml
Specifies the location of your servletcache.xml file.

Example:
`com.ibm.ws390.wc.config.dynsrvxmlfilename=/u/svr1/BBOASR4/servletcache.xml`

com.ibm.ws390.trace.settings=*path/file*
The fully qualified directory path and file name for the trace settings file.

For more information about trace settings, see:

- "Steps for preparing the z/OS or OS/390 environment for logging Java application messages and trace requests" on page 294, and
- *WebSphere Application Server V4.0.1 for z/OS and OS/390: Messages and Diagnosis*, GA22-7837.

Example: `com.ibm.ws390.trace.settings=/mydir/trace.settings`

com.ibm.ws390.wc.config.filename=*path/file*
The fully qualified directory path and file name for an optional file for Web container configuration information. Specify this property only if you want to override the default `webcontainer.conf` file.

Example:
`com.ibm.ws390.wc.config.filename=/usr/lpp/WebSphere/AppServer/bin/WebCon2.conf`

com.sun.tools.javac.main.largebranch=[true | false]
Enables JSPs to have up to 32K branches for internal compilation of Java files. If this property is set to false, only 16K branches for internal compilation of Java files are allowed.

| **invocationCacheSize=<size of the cache>**
| The invocation cache holds information for mapping request URLs to servlet
| resources. A cache of the requested size is created for each worker thread that
| is available to process a request.

| The number of worker threads is determined by the value specified for the
| BBOO_WORKLOAD_PROFILE environment variable. If more than 50 unique
| URLs (JSPs) are actively being used, you should increase this parameter.
| Increasing this parameter enables a cached object to be used. This avoids the
| need to create a new java object, and results in improved performance.

| **Note:** A larger cache uses more of the Java heap, so you might need to
| increase maximum Java heap size. For example, if each cache entry
| requires 2KB, maximum number of worker threads size is set to 25, and
| the URL invocation cache size is 100, you must allow for 5MB of heap
| storage that will be dedicated to this cache and inaccessible by the
| application.

mail.debug=[true | false]
Enables or disables JavaMail's debugging capability. When debugging is enabled, JavaMail will print out step-by-step mail-related interactions to stdout.

WebSphere for z/OS normally redirects messages to stdout.log and stderr.log files.

Note: The mail.debug property setting is shared by all mail session instances within the same process JVM. Therefore, debugging will be turned on for all mail session instances even if you only want to debug one of them.

org.omg.CORBA.ORBInitialHost=*host_name*
The Internet Protocol name that a z/OS or OS/390 client, or server region acting as a client, uses to access the bootstrap server (that is, when the client or server region invokes the resolve_initial_references method). The default is the value specified by the RESOLVE_IPNAME environment variable, which is the Internet Protocol name associated with the System Management Server (the default bootstrap server). If RESOLVE_IPNAME is not set, the value is the system on which the client or server region is running.

Use this property to specify a bootstrap server running on a remote system.

Note: The TCP/IP port number for org.omg.CORBA.ORBInitialHost is defined by org.omg.CORBA.ORBInitialPort.

The value of org.omg.CORBA.ORBInitialHost can be up to 255 characters.

Example:

```
org.omg.CORBA.ORBInitialHost=MYHOST.COM
```

org.omg.CORBA.ORBInitialPort=*host_port*
Specifies the TCP/IP port number for the host name specified on org.omg.CORBA.ORBInitialHost. Use this property to specify a bootstrap server running on a remote system.

The default is 900.

Example:

```
org.omg.CORBA.ORBInitialPort=1900
```

| **service.debug.enabled=[true | false]**
| Specifies whether or not additional debugging and tracing information, such as
| classpath, time to compile the source, and the version of the compiler, are to be
| included in the log file.

| The default is false.

| **Example:**

```
service.debug.enabled=fa
```

WEB_SECURITY_VERSION=1|2
Specifies which version of the Web Security Collaborator is to be used to provide Web container security.

Version 1 of the Web container security collaborator uses a SAF user registry and only provides the following security functions for requests received by the IBM HTTP Server for z/OS and forwarded to the Web container via the WebSphere for z/OS Local Redirector plug-in. None of these functions were available for requests received by the HTTP or HTTPS Transport Handlers:

- Basic authentication
- Form-Based authentication
- Client Certificates
- Single Sign-On across WebSphere/390 Servers

Version 2 of the Web container security collaborator enables the Web container to provide most of these same security functions for requests that are received by the HTTP or HTTPS Transport Handler as well as for requests received by the IBM HTTP Server for z/OS. This version of the collaborator is required if you are using a trust association interceptor or a custom user registry with WebSphere for z/OS.

Example:

```
WEB_SECURITY_VERSION=2
```

Appendix B. Default webcontainer.conf file

The following is a copy of the Web container configuration file, webcontainer.conf file. This copy 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.

```
####################################################################
# (C) COPYRIGHT 2001 IBM Corporation. All rights reserved.
#
appserver.version=4.01
# ================================================================== #
#
#  Configuration file for an IBM WebSphere Application Server
#  for z/OS and OS/390 version 4.0 Web container.
#
#  The documentation in this file provides descriptions of the properties
#  contained in the webcontainer.conf file.
#
#  NOTES ON SYNTAX:
#
#  The property names consist of fixed portions (e.g. host)
#  and variable portions (e.g. <virtual-hostname>).  The fixed portions
#  must be in lowercase; the variable portion can be in
#  mixed case and is case sensitive.
#
#  In the following example, host, and alias are fixed
#  portions of the property name and must be in lowercase, while
#  <virtual_hostname>, and <hostname> are the variable portions
#   within the property name and can be specified in mixed case.
#
#  ex. host.<virtual-hostname>.alias=<hostname>
#
#
# ================================================================== #
#
#        PROPERY GROUPINGS
#        =================
#        - Session
#        - Virtual Host
#        - Web Security
#
#        Note: Throughout this file, <applicationserver_root> refers
#        to the directory path of the mounted install image of the
#        product. The default is /usr/lpp/WebSphere.
#
#
# ================================================================== #
#
#   Session Settings
#
# ================================================================== #
#
#      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 Manager recognizes
#        session IDs that arrive in the URL and rewrites
#        the URL, if necessary, to send the session IDs.
#
#        The default is false.
#
#
session.urlrewriting.enable=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.cookies.enable=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.protocolswitchrewriting.enable=false
#
#--------------------------------------------------------------------#
#
#      session.cookie.name=<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 "JSESSIONID".
#
#
session.cookie.name=JSESSIONID
#
#--------------------------------------------------------------------#
#
#      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.comment=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.maxage=-1
#
#--------------------------------------------------------------------#
#
#      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.secure=false
#
#--------------------------------------------------------------------#
#
#      session.tablesize=<integer>
#
#        Specifies the size of the session table used to maintain
#        session objects within the Web container. When
#        session.tableoverflowenable=false, this is the limit on
#        the number of session objects that can be created by the
#        Web container 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.tablesize=1000
#
#--------------------------------------------------------------------#
#
#      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.
#
#        Note: The HTTP session timeout settings specified in the
#        webcontainer.conf file can be overridden for a particular
#        Web application by adding the following tags to the the
#        web.xml file for that application:
#
#            <session-config>
#                 <session-timeout>x</session-timeout>
#
#        where x is the timeout value, in minutes, for that
#        application. You can also override the webcontainer.conf file
#        setting by specifying the new value in the "Session timeout"
#        field on the "General" tab for the Web application when you
#        use the WebSphere for z/OS AAT to deploy the Web application
#        on your WebSphere for z/OS system.
#
session.invalidationtime=180000
#
#--------------------------------------------------------------------#
#
```

```
#      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.tableoverflowenable=true
#
#--------------------------------------------------------------------#
#
#      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.dbenable=false
#
#--------------------------------------------------------------------#
#
#      session.dbtablename=<database-tablename>
#
#        Specifies the database table name to be used by the session
#        services when session.dbenable=true.
#
#        There is no default.
#
#
session.dbtablename=
#
#--------------------------------------------------------------------#
#
#      session.cookie.domain=<domain_name>
#
#        Specifies the domain name for which the session cookie is
#        valid.
#
#        The default is null.
#
#
session.cookie.domain=
#
#--------------------------------------------------------------------#
#
#      session.reaperinterval=<integer>
#
#        Specifies the interval, in seconds, at which the reaper
#        (i.e, invalidator) will run. If this property is not
#        included in the webcontainer.conf file, or if the specified
#        value is less than 30, the interval will be automatically
#        caclulated.
#
#        The default value is 0, which enables the Web
#        container to automatically calculate the interval.
#
#
session.reaperinterval=0
#
#--------------------------------------------------------------------#
#
#      session.persistenceversion=1|2
#
```

```
#         Specifies which DB2 table format is to be used for storing
#         data.
#
#         If 2 is specified, the table format for DB2 Session
#         Persistence Version 2 must be used for your DB2 database.
#         This format is described in the section "Steps for
#         configuring HTTP Session Support" in "WebSphere Application
#         Server for z/OS and OS/390: Assembling J2EE Applications".
#         This table requires that an HTTP(S)Transport Handler is being
#         used to handle HTTP(S) requests to the Web container and
#         that a Web server plug-in has been configured for use with
#         WebSphere for z/OS.
#
#         If 1 is specified, the table format for DB2 Session
#         Persistence Version 2 must be used for your DB2 database.
#         This format is described in the section "Steps for
#         configuring HTTP Session Support" in "WebSphere Application
#         Server for z/OS and OS/390: Assembling J2EE Applications".
#         When 1 is specified, the WebSphere for z/OS local redirector
#         pl;ugin, as well as an HTTP(S) Transport Handler, can be
#         used to handle HTTP(S) requests.
#
#         Note: Values specified for the following session properties
#         will be ignored if this property is not set to 2:
#
#               session.timebasedwrite
#               session.timebasedwriteinterval
#               session.dbconnections
#               session.usingmultirow
#               session.writeallproperties
#               session.scheduledinvalidation
#
#         The default is 1.
#
#
session.persistenceversion=1
#
#---------------------------------------------------------------------#
#
#       session.timebasedwrite=true|false
#
#         Specifies whether or not database updates are done on a
#         separate thread. If true is specified, database updates are
#         done on a separate thread at the interval specified on
#         the session.timebasedwriteinterval property.
#
#         The true option gives better performance but leaves a
#         larger window for data loss during failover.
#
#         The default is false.
#
#
session.timebasedwrite=false
#
#---------------------------------------------------------------------#
#
#       session.timebasedwriteinterval=<integer>
#         Specifies the interval, in seconds, at which session updates
#         get written to the database by a background thread. This
#         property only appliesif the timebasedwrite property is set
#         to true.
#
#         The default is 120.
#
#
session.timebasedwriteinterval=120
#
#---------------------------------------------------------------------#
#
#       session.dbconnections=<integer>
#
#         Specifies the number of database connections that will be
#        held by the session manager for its exclusive use. Holding
#        connections will improve performance. By default, each JVM
#        runs three threads, in which case, specifying 3 DB2
#        connections is optimal.
#
#        The default is 0.
#
#
session.dbconnections=0
#
#--------------------------------------------------------------------#
#
#      session.usingmultirow=true|false
#
#        Specifies how session attributes are to be written:
#
#        1.  If true is specified, each session attribute will be
#        written in a separate row.
#        2.  If false, is specified, all session attributes will be
#        written in a single row.
#
#       If your session is used only for a few small attriubutes,
#       single-row support will probably result in better performance.
#
#       The default value true.
#
#
session.usingmultirow=true
#
#--------------------------------------------------------------------#
#
#      session.writeallproperties=true|false
#
#        Specifies that all properties are to be written to the
#        database even if they haven't been changed by a call to
#        setAttribute. This forces changes to objects previously
#        added to the session to be written to the database even if
#        setAttribute isn't called every time. False is the default
#        and will yeild better performance.
#
#        The default is false.
#
#
session.writeallproperties=false
#
#--------------------------------------------------------------------#
#
#      session.scheduledinvalidation=true|false
#
#        Specifies whether or not sessions are to be invalidated at
#        the two specific hours of the day that are specified on the
#        session.scheduledhour1 and session.scheduledhour2 properties.
#        This minimizes database access for invalidations, but
#        effectively means sessions will not timeout when the
#        interval specified on the MaxInactiveInterval property is
#        reached. Specifiying true should result in better performance.
#
#        The default is false.
#
#
session.scheduledinvalidation=false
#
#--------------------------------------------------------------------#
#
#      session.scheduledhour1=<integer_between_0_and_22>
#
#        Specifies the first hour of the day (0-22) at which
#        invalidation will occur if the scheduleinvalidation property
#        is set to true. Any value specified for this property is
#        ignored if the  scheduleinvalidation property is set to false.
#
#        The default is 0.
```

```
#
#
session.scheduledhour1=0
#
#--------------------------------------------------------------------#
#
#      session.scheduledhour2=<integer_between_1_and_23>
#
#        Specifies the second hour of the day (1-23) at which
#        invalidation will occur if the scheduleinvalidation property
#        is set to true. Any value specified for this property is
#        ignored if the scheduleinvalidation property is set to false.
#
#        The default is 1.
#
#
session.scheduledhour2=1
#
# ================================================================== #
#
#   Virtual Host settings
#
# ================================================================== #
#
#      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 HTTP server and the virtual host
#        definitions in the Application Server.
#
#        You can specify multiple aliases for the same virtual host on the
#        same property. To specify multiple aliases, separate each alias
#        name with a comma and a space. For example:
#        Host.default_host.alias=www.hostname.alias1, www.hostname.alias2,
#        www.hostname.alias3
#
#        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.
#
#        Note: If you are using the IBM HTTP Server as your HTTP protocol
#        catcher, you must make sure that the values specified for the  #
#        host.default_host.alias property in the
#        webcontainer.conf file match the values specified on the
#        host.<virtual-hostname>.alias
#        property in the V4.0.1 was.conf file,
#        which is initially set to localhost.
#
#        There is no default.
#
#
host.default_host.alias=localhost
#
#--------------------------------------------------------------------#
#
#      host.<virtual_hostname>.contextroots=<contextroot>[,
#              <contextroot>]
#
#        Binds installed Web applications into a specific virtual
#        host. The context root specified here corresponds to the
#        context root bound to the Web application during application
#        deployment.
#
#        One or more context root values can be specified. When
#        specifying multiple context root values, separate each value
#        with a comma. One or more spaces between values are permitted.
#
#        For example:
#          host.default_host.contextroot=/webapp/examples,  /payroll
#
```

```
#        The context root values specified can either be an explicit
#        match to the context root of the deployed web application
#        or you canuse a generic pattern. There are two types of
#        generic patterns:
#
#        1. "/"  which is a catch all context root. All Web application
#        context roots will match this pattern unless there is a
#        more specific context root defined.
#
#        If you are only configuring one virtual host definition in a
#        J2EE server, you can map all Web applications to that
#        virtual host definition by specifying a context root binding
#        of "/".   This will allow you to deploy future Web applications
#        into the server without having to update this property.
#
#        For example:
#           e.g. host.default_host.contextroots=/
#
#        2) "/appl/*" which is a generic pattern that means that any
#        Web application context root that begins with "/appl" will
#        match this pattern.
#
#        For example, the following context roots would all
#        match this pattern:
#          /appl
#           /appl/payroll
#           /appl/hr/benefits
#
#
#        Note: the use of "*" is limited to the last position in a
#        context root pattern and must be immediately proceeded by
#        a forward slash.
#
#        The following are examples of valid generic patterns:
#           /appl/*
#           /appl/payroll/*
#
#        The following are examples of invalid context root
#        patterns using *
#           *
#           /*
#           /appl*
#           /*/payroll
#
#       The rules for matching a Web application context root to a
#       virtual host context root pattern is as follows:
#
#       1. Find an exact match.
#
#       An exact match will always take precedence over a
#       generic pattern match. For example, if the following
#       context roots are specified for a virtual host:
#
#           host.vh1.contextroots=/webapp/examples/*
#           host.vh2.contextroots=/webapp/examples
#
#       A Web application with a context root of /webapp/examples
#       will be bound to virtual host name vh2, because it is an
#       exact match.
#
#       2. Find the pattern that matches best.
#
#       When multiple generic patterns match a Web application's
#       context root, the generic pattern that matches the most
#       qualifiers of the URI, starting from the left, is
#       considered the best match. For example, given the following
#       context roots:
#
#           host.vh1.contextroots=/webapp/examples/*
#           host.vh2.contextroots=/webapp/*
#           host.vh3.contextroots=/
#
```

```
#        A Web application context root of /webapp/examples/test
#        would be bound to virtual host vh1.
#        A Web application context root of /webapp/test
#        would be bound to virtual host vh2.
#        A Web application context root of /test
#        would be bound to virtual host vh3.
#
#         The default is "/".
#
#
host.default_host.contextroots=/
#
#--------------------------------------------------------------------#
#
#      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:
#        <applicationserver_root>/AppServer/bin/default_mimetype.properties
#
#
host.default_host.mimetypefile=
#
## ================================================================== #
#
#   Web Security Settings
#
#--------------------------------------------------------------------#
#
#   WebAuth.UnauthenticatedUserSurrogate=userid
#
#      Specifies a value that indicates either the SAF UserID or
#      CustomUserRegistryuserid under which unauthenticated clients are to
#      be executed.
#
#      If a SAF userid is going to be specified, the JVM property
#      WEB_SECURITY_VERSION must be set to 1.
#
#      If a CustomUserRegistry userid is going to be specified:
#      1. The JVM property WEB_SECURITY_VERSION must be set to 2.
#      2. The custom user registry implementation class must be
#         included on the WS_EXT_DIRS environment variable.
#
#***************************************************************************
#   WebAuth.CustomRegistry.ImplClass= #      com.company_name.implementation_class_name
#
#      Specifies the implementation class for the CustomUserRegistry interface.
#
#   Example:
#   WebAuth.CustomRegistry.ImplClass=com.ibm.websphere.security.CustomRegistry.
#
#***************************************************************************
#
#   WebAuth.CustomRegistry.Properties=filename.properties
#
#      Specifies the fully qualified name of the configuration file used
#      by the CustomRegistry interface
#      to setup its environment.
#
#   Example: WebAuth.CustomRegistry.Properties=
#              /u/MyCompany/security/cur.properties.properties
#
#   There is no default value for this property.
#
#****************************************************************************
#
#   WebAuth.CustomRegistry.authorizationTableXML=filename.xml
#
#     Specifies the fully qualified name of the XML file containing
#     the list of XML files containing authorization tables. If a J2EE server is
#     going to be using a custom user registry to authenticate and authorize
#     requests, every Web application installed on that  server must have
#     an entry in one of these tables. Use the following sample file to create
#     this type of file:
#
#   <AuthTableList>
#        <application>
#           <name>application_name</name>
#           <permission-file-name>fully_qualified_filename
#           </permission-file-name>
#        </application>
#        <application>
#           <name>application_name</name>
#           <permission-file-name>fully_qualified_filename
#           </permission-file-name>
#        </application>
#     </AuthTableList>
#
#      Include the following set of tags if you want to set up a default
#      authorization table:
#
#      <application>
#         <name>*</name>
#         <permission-file-name>fully_qualified_filename
#         </permission-file-name>
#      </application>

#   Example: WebAuth.CustomRegistry.authorizationTableXML=
#             /u/MyCompany/permissions.xml
#
#****************************************************************************
#
#   WebAuth.CustomRegistry.SAFPrincipal=MVS_ID
#
#     Specifies the valid MVS user ID under which requests from custom
#     registry clients are processed. (For example, this ID will be used when
#     processing EJB requests or to access an MVS resource that has a
#     deployment descriptor containing the attribute res-auth=container.) This
#     property is only required if you do not want to use the J2EE server's
#     ID to establish these connections.
#
#   Example: WebAuth.CustomRegistry.SAFPrincipal=Admin4
#
# ================================================================== #
#
#      WebAuth.LoginToken.Expiration=<minutes>
#
#      Specifies the number of minutes for which a login is valid.
#      When the token expiration period is reached, the user is forced
#      to authenticate again.  The value specified for <minutes>
#      must be a positive integer.
#
#      The default is 10 minutes.
#
#
WebAuth.LoginToken.Expiration=10
#
#--------------------------------------------------------------------#
#
#      WebAuth.LoginToken.LimitToSecureConnections=true|false
#
#      Specifieds a boolean value that, when set to true, indicates a transport
#      constraint is to be used for requests that require use of a login token.
#
#      If true is specified, WebSphere for z/OS will only return the cookie over
#      a secure connection.  Setting this value to true also instructs WebSphere
#      for z/OS to set the "secure" bit in the cookie containing the Login Token.
#      Setting the "secure" bit in the cookie instructs HTTP Clients, such as a
#      browser, to only send the cookie on requests that are being send on a
#      secure transport.
#
#      The default is true.
```

```
#
#
WebAuth.LoginToken.LimitToSecureConnections=true
#
#--------------------------------------------------------------------#
#
#      WebAuth.LoginToken.Encrypt=true|false
#
#      Specifies a boolean value that, when set to true, indicates that the
#      Login Token is to be encrypted.#
#
#      The default is true.
#
#
WebAuth.LoginToken.Encrypt=true
#
#--------------------------------------------------------------------#
#
#      WebAuth.EncryptionKeyLabel=<label_name>
#
#      Specifies the label of the cryptographic key (preferably a
#      triple-DES key) contained in the ICSF CKDS (Cryptographic Key
#      Data Set) that is to be used for Web application security.
#
#      There is no default value for this property.
#
#
WebAuth.EncryptionKeyLabel=
#
#--------------------------------------------------------------------#
#
#      WebAuth.SingleSignOn.Enabled=true|false
#
#      Specifies a A boolean value that, when set to true, indicates that
#      a Login Token can be used for multiple applications existing on different
#      WebSphere Application Server's serving as virtual hosts, provided these
#      virtual hosts reside in the domain specified in the
#      WebAuth.FormBasedLogin.SingleSignOnDomain property.
#
#      The default is false.
#
#
WebAuth.SingleSignOn.Enabled=false
#
#--------------------------------------------------------------------#
#
#      WebAuth.SingleSignOn.Domain=<domain_name>
#
#      Specifies the name of the domain to which a single sign-on is restricted.
#      This domain name is used when creating HTTP cookies for single sign-on,
#      and determines the scope to which single sign-on applies.
#
#      For example, a value of  austin.ibm.com would allow single sign-on to work
#      between WebSphere Application Server A with virtual host of
#      serverA.austin.ibm.com and WebSphere Application Server B with virtual host
#      of serverB.austin.ibm.com.
#
#      Note: Cross-domain Single Sign On is not supported. A server at
#      austin.lotus.com,and another at austin.ibm.com cannot partipicate
#      in single sign-on.
#
#      The default is NULL.
#
#
WebAuth.SingleSignOn.Domain=
#
# ================================================================== #
#
#      WebAuth.TrustAssociationInterceptor.<value>.ImplClass=<classname>
#
#      Specifies the implementation class for a trust association interceptor.
#      You must include one of these properties for each trust association
#      interceptor you will be using.
#        <classname> is the name of a trust association interceptor's
#            implementation class.
#        <value> is a unique string of alphanumeric characters that is used
#               to correlate a TAI with its property file. Even if a property file
#               is not being using, a character string, such as TA1, must be
#               included as a place holder.
#
#      Example: WebAuth.TrustAssociationInterceptor.TA1.ImplClass=class1
#
#         There is no default.
#
#
WebAuth.TrustAssociationInterceptor.<value>.ImplClass=
#
#-------------------------------------------------------------------#
#
#      WebAuth.TrustAssociationInterceptor.<value>.Properties=<filename>
#
#        This property is optional and is only required if the trust
#        association interceptor class is using a configuration file
#        to set up the environment for a trust association interceptor.
#
#         <value> is a unique string of alphanumeric characters that
#            is used to correlate this property file to a TAI.
#            This string must match the string specified on a
#            WebAuth.TrustAssociationInterceptor.
#            <value>.ImplClass property
#         <filename> is the name of the trust association interceptor's
#               properties file.
#
#      Example: WebAuth.TrustAssociationInterceptor.TA1.Properties=configFile1
#
#         There is no default.
#
#
WebAuth.TrustAssociationInterceptor.<value>.Properties=
#
# ================================================================= #
#
#   Migrating a Version 3.x was.conf properties file to
#   Version 4.0 webcontainer.conf
#
# ================================================================= #
#
#  The following V3.x properties can be used to
#  configure a V4.0 Web container. Update these properties, where
#  required, with environment-specific data and then uncomment the
#  properties and start the Application Server using these settings.
#
#  - Server properties: None. Server properties are established during the
#     J2EE server configuration process.
#
#  - Session properties
#
#      To start the Application Server with equivalent session support
#      configured, copy the following properties, with their current
#      settings, from your existing V3.x was.conf file to the new
#      V 4.0 webcontainer.conf file:
#
#            session.enable
#            session.urlrewriting.enable
#            session.cookies.enable
#            session.protocolswitchrewriting.enable
#            session.cookie.name
#            session.cookie.comment
#            session.cookie.maxage
#            session.cookie.secure
#            session.tablesize
#            session.invalidationtime
#            session.tableoverflowenable
#            session.dbenable
#            session.dbtablename
#            session.domain
#
#      The remaining V3.5 was.conf session properties are not supported for V4.0.
#
#  - Virtual Host properties
#
#      To start the Application Server with equivalent virtual host support
#      configured, copy the following properties, with their current
#      settings, from your existing V3.x was.conf file to the new
#      V 4.0 webcontainer.conf file:
#
##           host.<virtual-hostname>.alias
#            host.<virtual_hostname>.mimetypefile
#
#     Notes:
#        1. If you prefer, you can define a host called "default_host",
#           take the default mime types, and simply replace
#          <your-hostname> in the
#           host.default_host.alias=<your-hostname> property
#           with your specific hostname
#
#        2. 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.
#
#     Additionaly, you must provide a value for the new virtual host property,
#     host.<virtual_hostname>.contextroots, unless you want to use the
#     default value /.
#
######################################################################
```

Appendix C. Using the Alternate Configuration Option

You can install and configure the IBM HTTP Server to act as the HTTP catcher, with the V3.5 run-time environment serving as the execution environment for Web applications residing in the HTTP Server's address space. This configuration is referred to as the Alternate Configuration Option. Once you have configured your J2EE server environment, you can then also use the HTTP Server, along with the V3.5 run-time environment (or the WebSphere for z/OS local redirector plug-in, as it is sometimes called) to direct requests to Web applications residing in the J2EE server's Web container. However, at this point you should start using the HTTP Transport Handler to handle non-SSL requests to the Web container.

Notes:

1. During intialization of the Alternate Configuration, you might receive messages similar to the following:

   ```
   BossLog: { 0001} 2002/03/13 17:46:41.432 01 SYSTEM=IBIR CLIENT=OSD13015
         PID=0X01000046 TID=0X0AFCC578 0X000033  c=UNK
         ./bbocsess.cpp+4977 ... BBOU0638E Function read() failed with RV=0,
         RC=129, RSN=0594003D, .EDC5129I No such file or
         directory..hostname/ip:p30zos2.ibi.com

   BossLog: { 0002} 2002/03/13 17:46:41.436 01 SYSTEM=IBIR CLIENT=OSD13015
         PID=0X01000046 TID=0X0AFCC578 0X000033  c=UNK
         ./bboocomm.cpp+8992 ... BBOU0051E Internal communications error:
         REASON=C9C20CAE
         BossLog: { 0003} 2002/03/13 17:46:41.437 01 SYSTEM=IBIR CLIENT=OSD13015
         PID=0X01000046 TID=0X0AFCC578 0X000033  c=UNK
         ./bboosyse.cpp+749 ... BBOU0011W The function CORBA::throw_sysexcp(const
         char*,ULong,CompletionStatus)+748 raised CORBA
         system exception

         CORBA::COMM_FAILURE.  Error code is C9C20CAE.
   ```

   These messages occur because you have not defined a J2EE server. They can be ignored because you currently do not intend to use a Web container to host your Web applications.
2. As long as you continue using the IBM HTTP Server as your HTTP protocol catcher for any application installed in a Web container, you must make sure that the values specified for the **host.**<*virtual-hostname*>**.alias** properties in the V4.0.1 was.conf file match values specified on the **host.default_host.alias** property in the webcontainer.conf file. In the V4.0.1 was.conf file, this property is initially set to **localhost**.

Rules:

- For HTTP Servers to communicate with the J2EE servers, an instance of the WebSphere for z/OS Daemon (whose default start procedure is BBODMN) must be active on each system in the sysplex where the HTTP Servers are installed.
- To use a non-WebSphere for z/OS ORB in conjuction with the V3.5 runtime, such as the CICS ORB, the value of the appserver.java.extraparm=-DDisable.J2EE property in the V4.0 was.conf file must be set to True. However, be aware that changing the setting of this property removes your connection to the J2EE server environment from the V3.5 runtime. This means that any applications that are accessed using the HTTP Server must reside in the V3.5 runtime and not the Web container on the J2EE Server.

© Copyright IBM Corp. 2000, 2003

Applications residing in the Web container, must either be moved to the V3.5 run-time environment, or accessed using the HTTP Transport Handler instead of the HTTP Server.

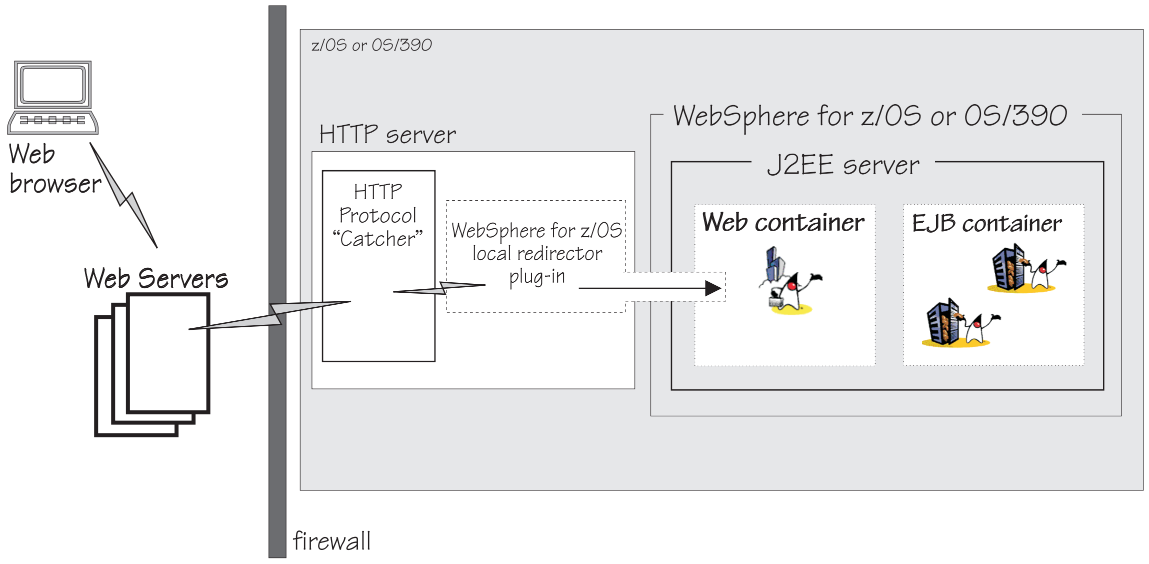


Figure 28. Alternative: Using an HTTP Server as the protocol catcher

Setting up the Alternate Configuration

When you install the WebSphere for z/OS product, the copy of the V3.5 Standard Edition run-time environment that is provided with the V4.0.1 product is placed in the IBM HTTP Server's address space. As shown in Figure 29 on page 367, locally residing Web applications (applications that reside within the V3.5 run-time environment) can be executed within this address space.

If you are not planning to initially utilize the capabilities of the full Java programming model, you do not have to configure a J2EE server environment at this time. You can use the configuration information contained in *WebSphere Application Server for OS/390 V3.5 Standard Edition Planning, Installing, and Using*, GC34-4835 to set up this V3.5 run-time environment to host and execute your Web applications.

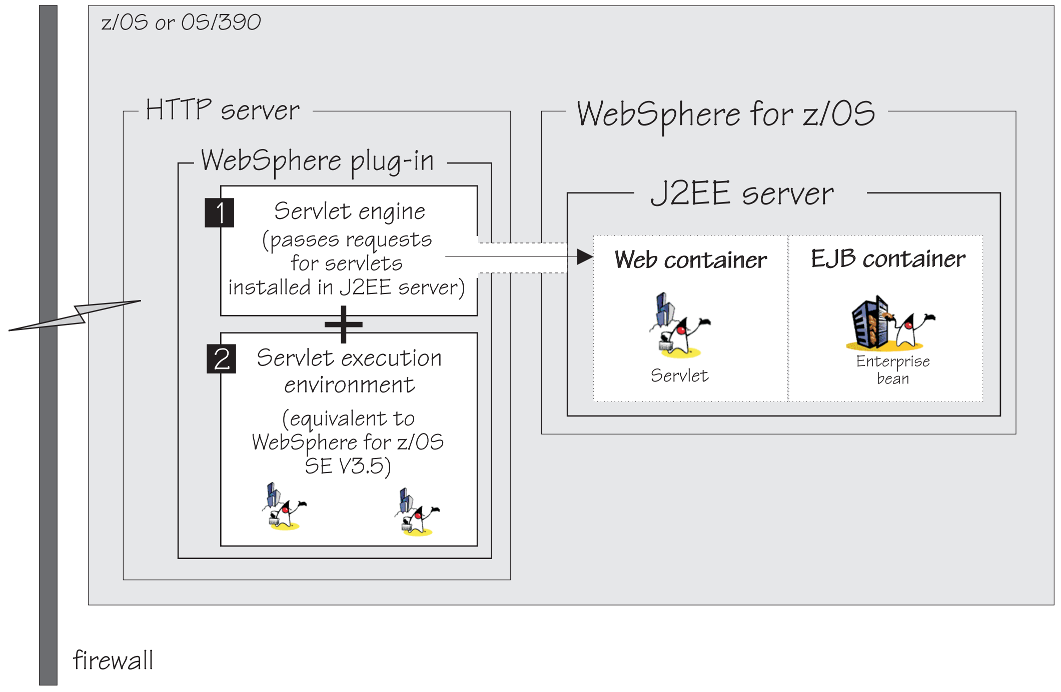

z/OS or OS/390
HTTP server
WebSphere plug-in
1
Servlet engine (passes requests for servlets installed in J2EE server)
2
Servlet execution environment
(equivalent to WebSphere for z/OS SE V3.5)
WebSphere for z/OS
J2EE server
Web container
EJB container
Servlet
Enterprise bean
firewall

Figure 29. Using Alternate Configuration Option

To correctly execute a request within this environment, the HTTP server, and the V3.5 run-time environment require specific configuration files that contain information you provide. These files are illustrated in Figure 30:
The following list summarizes, in general terms, the contents of those required

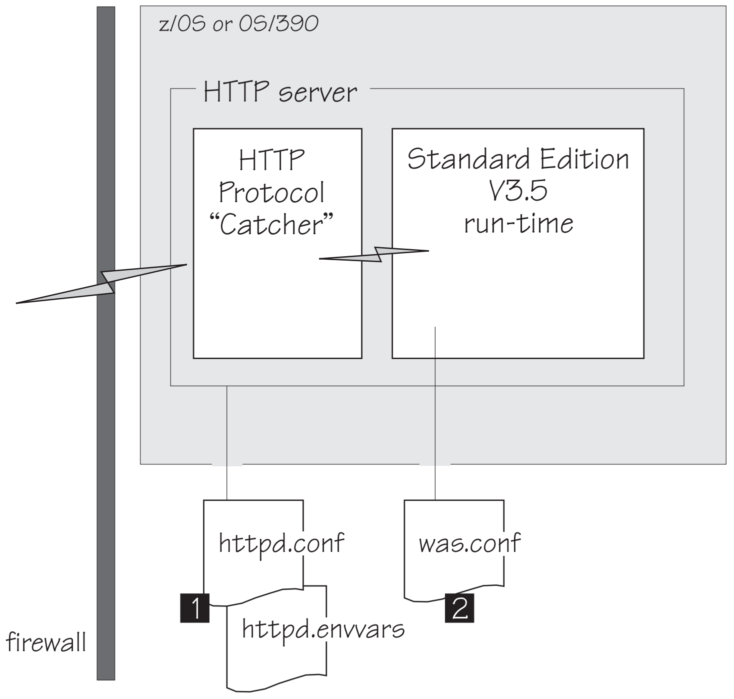


Figure 30. Files required for configuring the Alternate Configuration Option

configuration files and how they relate to the processing of inbound requests. More details are provided through an example in "Resolving requests to a specific servlet using the HTTP Server" on page 374.

1. The `httpd.conf` and `httpd.envvars` files allow the HTTP Server to interpret inbound requests for servlets.

   The `httpd.conf` file contains statements that allow the HTTP server to start the V3.5 run-time environment, and to send inbound requests to this environment for further processing. The `httpd.envvars` file contains environment variables that allow the V3.5 run-time environment to communicate with the HTTP Server. Although these variables have default values, the variable settings need to be reviewed to make sure the configuration is set up correctly.

   So, using the information in the `httpd.conf` and `httpd.envvars` files, the HTTP Server can catch inbound requests and passes them to the V3.5 run-time execution environment.
2. The V4.0.1 `was.conf` file should contains configuration and deployment information for Web applications that the V3.5 run-time environment is hosting. See *WebSphere Application Server Standard Edition V3.5: Planning, Installing and Using* for a description of the properties you need to include in the V4.0.1 `was.conf` file in order to execute these applications in this environment.

   If you are migrating from V3.5 Standard Edition and already have V3.5 Standard Edition `was.conf` file properties defining your Web applications:
   - Copy these webapp and deployedwebapp properties into the V4.0.1 `was.conf` file.
   - Install the Web applications into the HTTP Server address space where the V3.5 run-time environment provided with the V4.0.1 product has been installed.

"Resolving requests to a specific servlet using the HTTP Server" on page 374 provides an example of how the HTTP Server, and the V3.5 run-time environment work together to satisfy a sample application request.

When you are ready to start utilizing the full Java programming model, follow the instructions in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization* to configure your J2EE server. You can then install J2EE compliant Web applications in the Web container residing within the J2EE server and use either the HTTP Transport Handler described in "Using the HTTP Transport Handler configuration" on page 81, or the HTTP Server to dispatch requests to applications residing in the Web container. Requests to applications still residing in the V3.5 run-time environment can only be dispatched by the HTTP Server.

Instructions in Chapter 9, "Creating and running WebSphere for z/OS client applications," on page 217 explain how to enable Web applications residing in either the V3.5 run-time environment or the Web container to access Enterprise beans running in the WebSphere for z/OS J2EE server.

Steps for setting up an HTTP server

The IBM HTTP Server receives requests coming in from a network of browsers using the HTTP access protocol, and routes the request to the execution environment. You can either set up a new HTTP server to route requests to WebSphere for z/OS, or use an existing HTTP server. If you set up a new server to run alongside an existing one, you will probably need to pick a new TCP/IP port (such as 8080) at which the new server will be accessed.

Instructions for installing an HTTP server appear in *z/OS HTTP Server Planning, Installing, and Using*, SC34-4826. If you are setting up a new HTTP server, follow the instructions in that book, start the new server and verify its basic operations before proceeding with the instructions below. The procedure below only shows you how to modify an HTTP server for use with WebSphere for z/OS.

The HTTP server uses specific configuration files that need to be properly placed in the HFS. The recommended configuration structure looks like this:

```
/(root)
   /etc
       /Webserver_name
           httpd.conf (file)
           httpd.envvars (file)
```

where *Webserver_name* is the name of the HTTP server.

Perform the following steps to modify an existing HTTP server to route requests to WebSphere for z/OS:

1. Locate the `httpd.conf` and `httpd.envvars` configuration files for the HTTP server.

---

2. In the `httpd.conf` file, add the following statements:
   - The statement that identifies the initialization module for the V3.5 run-time environment:
     ```
     ServerInit /usr/lpp/WebSphere/WebServerPlugIn/bin/was400plugin.so:init_exit
     /usr/lpp/WebSphere,/dir_path/was.conf
     ```
     Replace the variable *dir_path*, with the HFS location of the `was.conf` file you want to use. Instructions for setting up that file appear in "Steps for configuring the V3.5 run-time provided with WebSphere for z/OS" on page 371..

     Note: If you do not specify a `was.conf` file on the `ServerInit` statement, WebSphere for z/OS uses the default `was.conf` file that is shipped with the product in the *`/usr/lpp/WebSphere`*`/WebServerPlugIn/properties` directory.
   - The service directive that identifies the module for processing inbound requests:
     ```
     Service /servlet/* /usr/lpp/WebSphere/WebServerPlugIn/bin/was400plugin.so:service_exit
     ```
     Replace the variable */servlet* with the context root for an individual servlet. You need to define one `Service` statement for each context root that is specified for modules in Web application to be installed in the J2EE server's Web container.
   - The statement that identifies the termination module for the V3.5 run-time environment:
     ```
     ServerTerm /usr/lpp/WebSphere/WebServerPlugIn/bin/was400plugin.so:term_exit
     ```

   Tips:
   - The variable */usr/lpp/WebSphere* represents the default location (HFS root) where WebSphere Application Server V4.0.1 is installed. Replace it with the correct root if the product is installed somewhere other than the default location. If you still have WebSphere Application Server Standard Edition Version 3.5 installed on the same system, make sure this root value does not point to the root for the Standard Edition product.
   - Be careful with upper- and lowercase letters in the samples below. Make sure you use uppercase letters exactly as shown.

---

3. In the `httpd.envvars` file, define the following variables:

   JAVA_PROPAGATE
   Set the value to NO if you will be using the HTTP Server, in conjunction with the V3.5 run-time environment provided with V4.0.1, to connect to Web applications in a J2EE Web container. Example:

   Example:JAVA_PROPAGATE=`NO`

   JAVA_HOME
   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.

   Example: JAVA_HOME=`/usr/lpp/java/IBM/J1.3`

   LIBPATH
   Add the reference to the exact location where the WebSphere for z/OS libraries are installed on your system.

   Example: `/usr/lpp/WebSphere/lib`

   NLSPATH
   Append the V3.5 run-time environment's message catalog directory *`applicationserver_root`*`/AppServer/Msg/%L/%N` to the existing NLSPATH statement.

   Example: If the existing NLSPATH setting is `/usr/lib/nls/msg/%L/%N:/usr/lpp/internet/%L/%N` and the V3.5 run-time environment is installed in `/usr/lpp/WebSphere`, then change the NLSPATH setting to: `/usr/lib/nls/msg/%L/%N:/usr/lpp/internet/%L/%N:` `/usr/lpp/WebSphere/AppServer/Msg/%L/%N`

   RESOLVE_IPNAME
   (Optional) After you have set up a J2EE server with a defined Web container, define the V3.5 run-time host you want to use to connect the V3.5 run-time environment to the J2EE server. This connection enables requests to pass through the HTTP Server to the Web container in the J2EE server.

   Example: `RESOLVE_IPNAME=wslx.washington.ibm.com`

   RESOLVE_PORT
   (Optional) Define the port that the V3.5 run-time environment will use to connect to the WebSphere for z/OS J2EE server.

   Example: `RESOLVE_PORT=900`

   Recommendation: These `RESOLVE_IPNAME` and `RESOLVE_PORT` variables have default values ("localhost" on port 900) that should suffice if the V3.5 run-time environment and WebSphere for z/OS run-time run on the same z/OS or OS/390 image, so specifying values for these variables is optional. To avoid any potential ambiguity or communication problems, however, code values for these two variables in the `httpd.envvars` file.

Now you are ready to complete the next procedure, "Steps for configuring the V3.5 run-time provided with WebSphere for z/OS" on page 371.

You also can configure session support through the HTTP server. Session support is a method of tracking information received during a session (that is, a series of requests that originate from the same user at the same browser). For further information, see "HTTP session support" on page 87.

Steps for configuring the V3.5 run-time provided with WebSphere for z/OS

The Version 3.5 run-time environment provided with WebSphere for z/OS acts as the server engine for the Web-serving environment. This run-time environment can route requests to the J2EE server's Web container for remote execution, but should be used primarily to run servlets locally in the HTTP Server's address space. In the latter case, the V3.5 run-time environment serves as both the servlet engine and servlet execution environment.

Through the following procedure, you will add configuration files to the HFS directories that the HTTP Server uses. When you have finished the steps below, the configuration structure will look like this:

```
/(root)
   /etc
       /Webserver_name
           httpd.conf (file)
           httpd.envvars (file)
           was.conf (file)
    /var
       /Webserver_name
           was_logs <dir>
           work <dir>
```

Perform the following steps to configure the V3.5 run-time environment:

1. In the HFS, create a working directory and a log directory for the V3.5 run-time environment, with permission bits 777.

   Example: You can use names like the following for these directories:

   ```
   /var/Webserver_name/work
   /var/Webserver_name/was_logs
   ```

   where *Webserver_name* is the name of the HTTP Server that you started using the instructions in "Defining the server configuration" on page 149.

   ____

2. Edit the default `was.conf` file provided with WebSphere for z/OS. The default `was.conf` file is located in the `/usr/lpp/WebSphere/WebServerPlugIn/properties/` directory.
   - Update the `appserver.workingdirectory` and `appserver.logdirectory` properties to specify the working and log directories that you created in the previous step.
   - If you have an existing `was.conf` file that you previously used for WebSphere Application Server Standard Edition Version 3.5, copy the following properties into the default `was.conf` file:
     - `deployedwebapp` and `webapp` properties defining any Web applications that you want to run in the V3.5 run-time environment instead of in the Web container.
     - `host` properties defining any virtual hosts you want to continue using.
   - Add the following versions of the `appserver.java.extraparm` property to the was.conf file if you do not want to use the default values:

`appserver.java.extraparm=-Dcom.ibm.ws390.wc.includedWebContainers=`
`<server1>,<server2>,...`
This parameter is used to specify the installed J2EE servers with which the WebSphere for z/OS V3.5 runtime can communicate. The servers in the list are seperated by a comma. If this parameter is not specified, the V3.5 runtime is able to communicate with all of the installed J2EE servers.
Example: In this example, only the BBS1104 J2EE server can communicate with the V3.5 runtime.
appserver.java.extraparm=-Dcom.ibm.ws390.wc.includedWebContainers=BBS1104

`appserver.java.extraparm=-Dcom.ibm.ws390.wc.serverCheckInterval=`
`<interval>`
This parameter is used to indicate how frequently, in minutes, the WebSphere for z/OS V3.5 runtime should check the list of J2EE servers specified on the `-Dcom.ibm.ws390.wc.includedWebContainers` parameter to determine if new servers have been added. If no value is specified for this property, the V3.5 runtime will check for new servers every 10 minutes.
Example: In this example, the V3.5 runtime will check for the specification of new J2EE servers every 15 minutes.
appserver.java.extraparm=-Dcom.ibm.ws390.wc.serverCheckInterval=15

`appserver.java.extraparm=-Dcom.ibm.ws390.wc.webappupdateInterval=`
`<interval>`
This parameter is used to indicate how frequently, in minutes, the WebSphere for z/OS V3.5 run-time should check to see if any new Web applications have been installed on the J2EE servers to which it dispatches requests. If no value is specified for this property, the V3.5 runtime will check for new Web applications every 2 minutes.
Example: In this example, the V3.5 runtime will check for new Web applications every 4 minutes
appserver.java.extraparm=-Dcom.ibm.ws390.wc.webappupdateInterval=4
For more information about the `appserver.java.extraparm` property, see "Template for the WebSphere for z/OS plug-in was.conf file" on page 376.

To test whether you correctly defined the V3.5 run-time environment's configuration files:
1. Restart the HTTP Server.
2. Browse the SYSOUT for the HTTP Server's started task for the following messages, which indicate that the V3.5 run-time environment started:
```
.....IBM WebSphere Application Server native plugin initialization went OK :-)
IMWO235I Server is ready.
```
The initialization process might take a few minutes to complete, even though the HTTP Server is operational.
3. After the V3.5 run-time environment has initialized, enter `http://host_name:host_port/webapp/examples/index.html` in a Web browser. In this URL, *host_name* and *host_port* are the TCP/IP nodename and port of the HTTP Server.

You should now see a screen with the WebSphere Application Server logo and the heading **Examples**. If you receive this screen, the HTTP Server successfully mapped your URL over to the V3.5 run-time environment, using the `Service` statements in the `httpd.conf` file. At this point, however, the V3.5 run-time environment served up this screen locally; no interaction with the V4.0.1 run-time has taken place.

You can now follow the procedures described in *WebSphere Application Server Standard Edition V3.5: Planning, Installing and Using* to install these applications into the V3.5 run-time environment.

Creating a customized JVM properties file for the Local Redirector Plug-in

For most environments, the JVM properties provided in the default_global.properties file are sufficient. However, if you want to use some of the advanced functions provided in the V4.0.1 environment, you can either add the **appserver.java.extraparm** properties described in the preceding section to your was.conf file, or create a customized JVM properties file. To create a customized JVM properties file:

1. Copy the default_global.properties file from the /usr/lpp/WebSphere/WebServerPlugIn/properties/ directory to the directory in which your plug-in's was.conf file resides. Make sure the file has permissions of at least 644.
2. Rename the copied file to something other than default_global.properties. It is no longer a ″default″ properties, so something like jvm.properties would be better.
3. Update the appserver.jvmproperties property in your was.conf file with the directory and filename of your copied and renamed file.
4. Add one or more of the following properties to your copied and renamed file:
   - com.ibm.ws390.wc.includeWebContainers=<server1>, <server2>, <server3>, ...

     Use this property to limit the J2EE servers with which the plug-in can communicate to those specified list of servers. This would be applicable in an environment where, for example, you had twenty application servers, one of which had web applications deployed and nineteen of which did not.

     For example, if you had a server named APSRV3, and that server had five instances defined as APSRV3S1, APSRV3S2, APSRV3S3, APSRV3S4, and APSRV3S5, then adding the following the property would limit the plug-in's scope to the Web containers in the five instances defined for the server APSRV3:

     ```
     com.ibm.ws390.wc.includWebContainers=APSRV3
     ```
   - com.ibm.ws390.wc.serverCheckInterval=<interval in minutes>

     If you want to alter the interval between which the plug-in checks for new J2EE servers. By default the plug-in will check every 10 minutes to see if additional J2EE servers have been defined. You may wish to set a longer time if your environment is relatively stable regarding the number of servers configured.
   - com.ibm.ws390.wc.webappupdateinterval=<interval in minutes>

     If you want to alter the polling interval used by the plug-in to check for new applications. By default, the plug-in will query each application server every two minutes to see if any new applications have been deployed. You may want to specify a higher value, particularly if your rate of new application introduction is low, or if you have a large number of servers and wish to minimize the amount of polling.
5. Stop and restart the Web server.

Resolving requests to a specific servlet using the HTTP Server

To understand how the HTTP server and the V3.5 run-time environment work together, and use information in configuration files to satisfy a servlet request, consider the following example. Suppose that your installation wants to do the following:

- Continue using a Web site that you previously set up for the state of Maryland police, fire, and tax authorities on WebSphere Application Server Standard Edition V3.5 system.

  Each of the three authorities has its own Web page and supporting Web applications. The left side of the figure depicts the three Maryland state Web pages: one for the fire authority, one for the police, and one for the tax authority. Under each is the URL that will appear on inbound servlet requests; the URL contains the domain name for each state authority page (the domain names are underlined in the figure). The URL also contains information that identifies the servlet to be run (this information is shaded in the figure).
- Host these different Web applications in the V3.5 run-time environment on WebSphere for z/OS.

  Although these Web applications will be installed in the same V3.5 run-time environment, your installation wants to keep the applications isolated, or logically grouped, by each state authority. The right side of the figure depicts the logical configuration that your installation wants to achieve. In this configuration, the V3.5 run-time environment has three logical partitions that correlate with the domain names for each Web page: One for the fire authority, one for the police, and one for the tax authority. These partitions are virtual hosts, which separate each authority’s Web applications from any other Web applications defined to this run-time environment. These virtual hosts also ensure that each authority’s servlets are run only in response to requests that contain the appropriate domain name.

The key to accomplishing these goals is to copy all of the properties defining the virtual hosts and Web applications for each of the three authorities from the Standard Edition V3.5 was.conf file to the V4.0.1 was.conf file.

Using the Maryland state tax authority Web page as a model, suppose that this tax authority page allows users to request information about income taxes, property taxes, or automobile taxes. In other words, inbound requests from the tax authority page, with domain name taxes.state.md.us, drive the GovTaxes application that includes the IncomeTax servlet, the PropertyTax servlet, or the AutoTax servlet. In this case, your installation needs to copy the following properties from the V3.5 was.conf file to the V4.01 file to configure this virtual host and the application containing these servlets to the V3.5 run-time environment:

```
host.taxes.alias=taxes.state.md.us
```

which defines the virtual host named taxes in the V3.5 run-time environment. (This virtual host correlates the taxes.state.md.us domain name on inbound requests.)

```
deployedwebapp.GovTaxes.host=taxes
```

which deploys this application under the taxes virtual host.

```
webapp.GovTaxes.servlet.IncomeTx.description=Income Taxes
webapp.GovTaxes.servlet.PropTx.description=Property Taxes
webapp.GovTaxes.servlet.AutoTx.description=Auto Taxes
```

which describe the servlets contained in this application, and any other deployedwebapp and webapp properties associated with the GovTaxes application.

With these properties in your V4.01 was.conf configuration file, you now have constructs that match key elements of the URL in inbound servlet requests from the tax authority Web page:

- The virtual host `taxes` matches the underlined domain name `taxes.state.md.us` in the figure
- The context roots `/auto`, `/property`, and `/income` match the shaded information in the figure.

The virtual hosts and application definitions determine how the V3.5 run-time environment handles inbound requests. Figure 31, and the following list illustrate the additional configuration information required to process an inbound request from the Maryland state tax page. Numbers in the list correspond to the numbers in the figure.

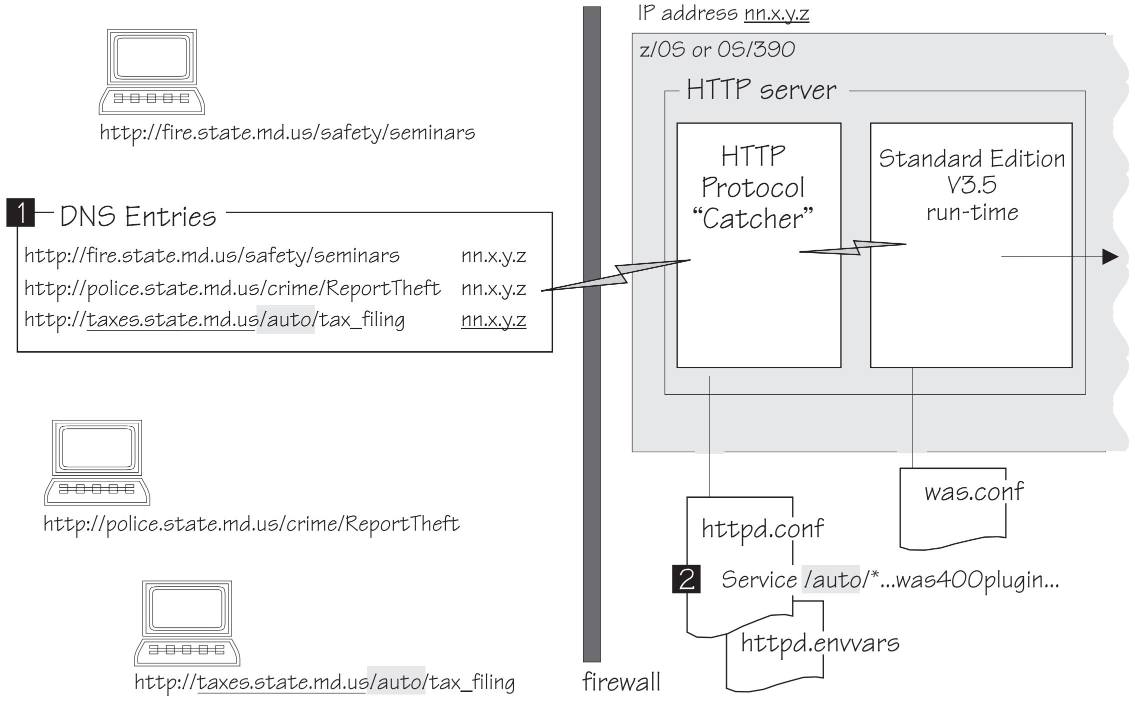

IP address nn.x.y.z
z/OS or OS/390
HTTP server
HTTP Protocol "Catcher"
Standard Edition V3.5 run-time
http://fire.state.md.us/safety/seminars
1 DNS Entries
http://fire.state.md.us/safety/seminars nn.x.y.z
http://police.state.md.us/crime/ReportTheft nn.x.y.z
http://taxes.state.md.us/auto/tax_filing nn.x.y.z
http://police.state.md.us/crime/ReportTheft
was.conf
httpd.conf
2 Service /auto/*...was400plugin...
httpd.envvars
http://taxes.state.md.us/auto/tax_filing
firewall

Figure 31. Routing an inbound request to the V3.5 run-time environment

1. A user browses the Web site for the Maryland state tax authority, and submits a request for information about automobile taxes. The inbound request (URL) is:`http://taxes.state.md.us/auto/tax_filing`

   The Domain Name Server directs the servlet request to the appropriate computer system: The Domain Name Server finds a DNS entry for `taxes.state.md.us`, and passes the request to IP address `nn.x.y.z.`, which is the adapter of the z/Series machine you want to use for Web-serving.
2. The HTTP Server recognizes the inbound request as one for the V3.5 run-time environment to process, and passes the inbound request to this environment.

   The HTTP Server at `nn.x.y.z.` catches the inbound request, and finds a matching `Service` statement in its `httpd.conf` configuration file. This `Service` statement contains the context root `/auto/*`, which matches part of the URL (`/auto`) for the inbound request. The `Service` statement also includes the HFS location and name of the V3.5 run-time module (`was400plugin`) for processing inbound requests.

The `httpd.envvars` file contains environment variables that allow the HTTP Server to communicate with the V3.5 run-time environment.
3. The V3.5 run-time environmente executes the application locally.

Note: After a J2EE server has been defined that contains a Web container, the V3.5 run-time environment uses the following technique to determine whether a given request is to be processed locally (the application resides in the same address space as the V3.5 run-time environment) or passed on to the Web container.

1. To find the appropriate execution environment, theV3.5 run-time environment first uses the `ServerInit` statement in the `httpd.conf` file to find the `was.conf` configuration file. It then scans that `was.conf` file for application information. If the V3.5 run-time environment finds matching application information, the WebSphere for z/OS plug-in runs the servlet locally; that is, the servlet runs within this environment, rather than being forwarded to the J2EE server's Web container for execution.
2. Otherwise, the V3.5 run-time environment assumes that the appropriate execution environment is the Web container in the WebSphere for z/OS J2EE server. It then uses the `jvm.properties` file to find the Web container's configuration file, to match the domain name on the inbound request to a virtual host defined to the Web container.

   In the configuration file, `webcontainer.conf`, the V3.5 run-time environment uses two key statements:
   - The `host.taxes.alias` statement, which equates the name of a virtual host to the URL host name on an inbound request (`taxes.state.md.us`), and
   - The `host.taxes.contextroots` statement, which binds specific servlets to this virtual host.

   At this point, the V3.5 run-time environment knows the inbound request is valid for Web container execution environment, because the domain name and application context in the request match a defined virtual host and its associated context roots, respectively. It then passes the request to the Web container in the J2EE server.

Template for the WebSphere for z/OS plug-in was.conf file

Following is a copy of the template of the WebSphere for z/OS V4.0.1 plug-in's was.conf file. 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 directory **/usr/lpp/WebSphere/WebServerPlugIn/properties**. For more information about how to use these properties, see *WebSphere Appliation Server Standard Edtion for OS/390 Version 3.5 Planning, Installing, and Using*.

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 4.01.
#
#  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 4.01.
#
#  - 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
#        =================
#        - Run-time Environment 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.
#
#
# ================================================================== #
#
#   Run-time Environment Settings
#          - Version
#          - Classpath/Libpath/Path settings
#          - JVM settings
#          - Logging level & location
#          - Working Directory
#          - Servlet 2.2 Compliance mode
#
# ================================================================== #
#
#
#      appserver.version=4.01
#
#        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.
#
#        A value of 4.0 or 4.01 MUST be specified for this property.
#
#
appserver.version=4.01
#
```

```
#--------------------------------------------------------------------#
#
#      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=/usr/lpp/ldap/lib/ibmjndi.jar<: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.permissions=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.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.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 of the "WebSphere Application Server Standard Edition
#        for OS/390 V3.5 Planning, Installing, and Using" publication
#        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 initialiation.  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 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=<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 "SESSIONID".
#
session.cookie.name=SESSIONID
#
#-------------------------------------------------------------------#
#       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.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.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
#
#        The pool name SessionJDBCConnectionPool is example.
#          Whatever name is used must match the value specified
#          in session.dbdbcpoolname.
#        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
#        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. The <string> value can be one of :
#
#        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 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.
#
#        Note: If you are using the IBM HTTP Server as your HTTP protocol
#        catcher, you must make sure that the values specified for the
#        host.<virtual-hostname>.alias
#        properties in the V4.0.1 was.conf file match values specified on
#        the host.default_host.alias property in the
#        webcontainer.conf file. In the V4.0.1 was.conf file, this property
#        is initially set to localhost.
#
#        This property is initially set to localhost.
#
#
#-------------------------------------------------------------------#
#
#      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 search 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 4.01
#
# ================================================================== #
#
```

```
#  The following are the minimal set of properties required to
#  configure an AppServer V4.01 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 4.01
#          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 4.01 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 4.01 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 4.01
#      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 4.01
#      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 4.01. It is recommended that you start with the default
#      logging properties of version 4.01 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 v4.01 Application Server, you need to consider
#      the following:
#
#      - JVM classpath - The version 4.01 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 4.01 appserver.classpath.
#      - Reloadable classpath - The version 4.01 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 4.01 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 4.01 Application
#            Server in anticipation of the possible need to
#            migrate either servlets or JSP to the new APIs
#            APIs supported in version 4.01.
#
#####  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 4.01
#      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 4.01 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 4.01 property
#         Version 4.01 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 4.01.
#
#      code -
#        version 1.x property
#          servlet.<servlet-name>.code=<servlet-class>
#        version 4.01 property
#
#      initargs -
#        version 1.x property
#          servlet.<servlet-name>.initArgs=<initargs>
#        version 4.01 property
#
#      autostart -
#        version 1.x property
#          servlets.startup=<servlet-name1> <servlet-name2> ...
#        version 4.01 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 4.01 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>
#
################################################################################
```

default_global.properties

Following is a copy of the default_global.properties file used with the V3.5 run-time environment provided with the WebSphere for z/OS product.

```
################################################################
# @(#)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=
#
##################################################################
```

Appendix D. Running the SOAP Installation Verification Program

Before you can run the SOAP Installation Verification Program:

1. If you are using the HTTP Transport Handler to handle application requests, go to step 2. If you are using a z/OS or OS/390 IBM HTTP Server to handle application requests, you must modify your HTTP Server's httpd.conf file to:
   - Include the following Service directive for the SOAP IVP:

     ```
     Service /soapivp/* /<install_root>/WebServerPlugIn/bin/was400plugin.so:service_exit
     ```
   - Set the Userid parameter to <*surrogate ID*> if it is currently set %%CLIENT%%:

     ```
     Userid <surrogate ID>
     ```
2. You must deploy the SoapIVP.ear file into the Web container on your J2EE server.

To deploy the SoapIVP.ear file:

1. Change directory to **/usr/lpp/WebSphere/samples**.
2. Find the SoapIVP.ear in the samples directory.
3. FTP the SoapIVP.ear to your workstation in binary format.
4. Start the WebSphere for z/OS Administration Application.
5. Select ″Add a conversation″ and add **SoapIVP**, and then save the conversation.
6. Expand the Tree: <sysplex_name> → <plex_name> → J2EE Servers →. You should see the J2EE Servers that are defined. Right click on the your J2EE Server and select the ″Install J2EE Application″.
7. Select the SoapIVP.ear file from your local file system and press ENTER.
8. Select the Soap-ivp-ejb bean, set the JNDI PATH to /soapivp/, and place AdderService in the JNDI Name field.
9. Set the soap_WebApp bean to the default JNDI name and path.
10. Click on the ″OK″ button. If the ″OK″ button remains grayed out and you cannot click on it, one of the required JNDI values has not been filled-in. Re-check these fields and ensure you have a green tick mark on each of the bean symbols in the SOAP IVP J2EE application's module tree.
11. Click on the ″OK″ button again. The Administration Applicaton will install the SOAP IVP as a J2EE application.

Once the SOAP IVP is activated:

1. Issue the following command to verify that the directory where the SoapIVP application is deployed has been created. If this directory was not properly created, you will receive an error message stating that the system cannot find the path specified.

   ```
   cd  <targetdir>/apps/<server_name>/SOAPIVP
   ```

   Note: If your <*targetdir*> is located on a read-only HFS, you will need to copy the SoapIVPClients jar file to a directory on an HFS to which you have write access, and then change to that directory, before continuing to the next step.

2. Change your directory to /usr/lpp/WebSphere/samples, and locate the SoapIVPClients.jar file. Issue the following command to unjar this file to the HFS containing WebSphere for z/OS:

   ```
   jar -xvf SoapIVPClients.jar
   ```
3. Issue the following command to list the current directory:

   ```
   ls
   ```
   You should see the following directory entries:

   ```
   com/ibm/soap/soapivpclients/
   ejbadder   scripts
   ```
   This directory contains the EJBAdderIVP.sh script that can be used to test your SOAP installation.
4. Make sure the Execute permission bit is set for this script.
5. From the command line, issue the following command to make sure JAVA_HOME is set to the exact location where the required level of the Software Development Kit (SDK) is installed on your system:

   ```
   echo $JAVA_HOME
   ```
   If JAVA_HOME is not set to the correct location, issue the following command to change its value:

   ```
   export JAVA_HOME=<SDK_install_root>
   ```
6. From the command line, issue the following command to make sure WAS_HOME is set to the correct value for your WebSphere for z/OS environment:

   ```
   echo $WAS_HOME
   ```
   If WAS_HOME is not set to your WebSphere for z/OS <*install_root*>, issue the following command to change its value:

   ```
   export WAS_HOME=<install_root>
   ```
   The default value for the WebSphere for z/OS <*install_root*> is **/usr/lpp/WebSphere**.
7. Start the J2EE Server.
8. From the command line, change the directory to **com/ibm/soap/soapivpclients/scripts**.
9. Then issue the following command to invoke the EJBAdderIVP.sh script, which is located in this directory:

   ```
   ./EJBAdderIVP.sh <server_location:port_number>
   ```
   This service:
   - Invokes the stateless Enterprise bean, AdderService.
   - Passes AdderService three integers to be added.
   - Returns the resulting sum, which should be 5.

   Note: The value specified for <*server_location:port_number*> must match the value of the virtual host to which this application is bound, which is specified in the webcontainer.conf file. If the value of the value for the virtual host is www.yourhost.com:8080, you would issue the following command:

   ```
   ./EJBAdderIVP.sh www.yourhost.com:8080
   ```

Appendix E. Notices

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

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

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

For license inquiries regarding double-byte (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

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

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

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

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

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
USA

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 information 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.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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

All statements regarding IBM’s future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

Examples in this book

The examples in this book are samples only, created by IBM Corporation. These examples are not part of any standard or IBM product and are provided to you solely for the purpose of assisting you in the development of your applications. The examples are provided ″as is.″ IBM makes no warranties express or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose, regarding the function or performance of these examples. IBM shall not be liable for any damages arising out of your use of the examples, even if they have been advised of the possibility of such damages.

These examples can be freely distributed, copied, altered, and incorporated into other software, provided that it bears the above disclaimer intact.

Programming interface information

This publication documents intended Programming Interfaces that allow the customer to write programs to obtain services of WebSphere for z/OS.

Trademarks

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

AIX	Open Class
CICS	OS/390
DB2	RACF
IBM	VisualAge
IMS	VTAM
IMS/ESA	WebSphere
Language Environment	z/OS

The term CORBA used throughout this book refers to Common Object Request Broker Architecture standards promulgated by the Object Management Group, Inc.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both. The Duke logo is a trademark or registered trademark of Sun Microsystems, Inc. in the United States and other countries.

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

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

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

Glossary

For more information on terms used in this book, refer to one of the following sources:

- Sun Microsystems Glossary of Java Technology-Related Terms, located on the Internet at:

  `http://java.sun.com/docs/glossary.html`
- *IBM Glossary of Computing Terms*, located on the Internet at:

  `http://www.ibm.com/ibm/terminology/`
- The Sun Web site, located on the Internet at:

  `http://www.sun.com/`

© Copyright IBM Corp. 2000, 2003

Index

Program Number: 5655-F31

Printed in the United States of America

SA22-7836-06