WebSphere® Application Server V4.0 for z/OS and OS/390

Assembling J2EE Applications

SA22-7836-01

WebSphere® Application Server V4.0 for z/OS and OS/390

Assembling J2EE Applications

SA22-7836-01

Note

Before using this information and the product it supports, be sure to read the general information under "Appendix D. Notices" on page 167.

Second Edition (June 2001)

This is a major revision of SA22–7836–00

This edition applies to WebSphere Application Server V4.0 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 for z/OS and OS/390 publications are at this Web site: `http://www.ibm.com/software/webservers/appserv/`

Contents

Figures

Tables

About this book

WebSphere Application Server V4.0 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 Application Server V4.0 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 for z/OS and OS/390," referred to in this text as "WebSphere for z/OS."

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, refer to the Sun Microsystems J2EE specification, which is available at:

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

Where to find related information

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

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

- *WebSphere Application Server V4.0 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 for z/OS and OS/390: License Information*, LA22-7855, describes the license information for WebSphere for z/OS.
- *WebSphere Application Server V4.0 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 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 for z/OS and OS/390: Operations and Administration*, SA22-7835, describes system operations and administration tasks.
- *WebSphere Application Server V4.0 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. It also includes information about migrating applications from previous releases of WebSphere Application Server for OS/390, or from other WebSphere family platforms.
- *WebSphere Application Server V4.0 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 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 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.

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/
```

Here are some books that you might find particularly helpful:

- *Getting Started with WebSphere Application Server*, SC09-4581, provides an overview of WebSphere for z/OS and describes requirements for setting up the environment.
- *Building Business Solutions with WebSphere*, SC09-4432

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 SA22-7836-01
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 17, 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 67 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 Application Server V4.0 for z/OS or OS/390: Enabling Web Applications on a J2EE server*:
  - “Enabling J2EE server support for Web applications (optional)” on page 47
  - “Appendix B. Default webcontainer.conf file” on page 153
  - “Appendix C. Migration considerations for Web applications running on WebSphere Application Server Standard Edition” on page 161
- The information in “Chapter 8. Creating and running J2EE application clients” on page 73 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 EditionWebSphere Application Server running on a workstation platform.
- The information in “Logging messages and trace data for Java applications” on page 87 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 93 also have changed to reflect the changed destinations for messages.

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

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

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. Figure 1 presents an illustration of the WebSphere for z/OS environment.

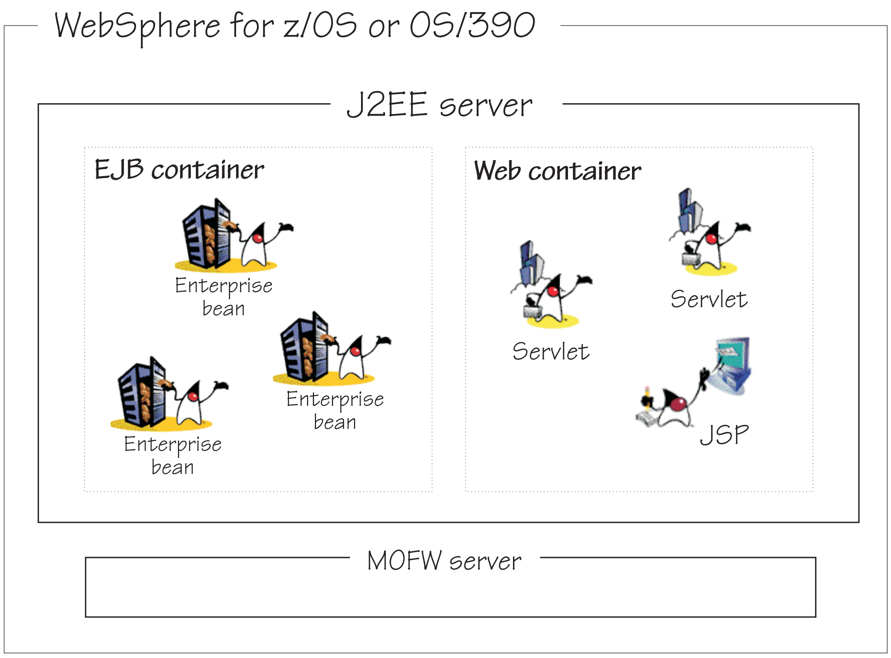

Figure 1. The new WebSphere for z/OS server for J2EE applications

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

Current WebSphere for z/OS support includes the J2EE technologies 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
Java 2 Standard Edition (J2SE) Software Development Kit (SDK)	V1.3
Enterprise JavaBeans	V1.1 (but also supports V1.0 specification)
Java servlets	V2.2 specification
JavaServer Pages (JSPs)	V1.1 specification
Java Transaction Service (JTS) and API (JTA)	V1.0 supported with distributed transactions
Java Database Connectivity (JDBC)	JDBC V2.1 and JDBC Standard Extensions V2.0 (JDBC V1.x is supported for compatibility)
Java Naming and Directory Interface (JNDI)	1.2
Java Remote Method Invocation (RMI) and RMI/IIOP	1.0
Java IDL	Supported
Java Messaging Service (JMS)	Not currently supported
JavaBeans Activation Framework (JAF)	Not currently supported
JavaMail	Not currently supported
Connectors	For Web components only: IBM Common Connector Framework (CCF) connectors: • The IMS Connect product (5655-E51) • CICS Transaction Gateway product (5648–B43)

The WebSphere for z/OS J2EE server supports a variety of client applications, which can access either Enterprise beans or Web applications, as shown in Figure 2 on page 5. Note that:

- Components in a WebSphere for z/OS J2EE server may be clients of components in another J2EE server, on the same or a different z/OS or OS/390 image. Also:
  - Servlets can drive CORBA-based Java business objects (BOs) in a MOFW server.
  - Enterprise beans and Java BOs can interoperate.

C++ business objects in a MOFW server, however, cannot access application components in a J2EE server.

- Web components (servlets and JavaServer Pages) that run in WebSphere Application Server Standard Edition can be:
  - Clients that drive Enterprise beans in the WebSphere for z/OS J2EE server.
  - Web components that can be migrated to run in the WebSphere for z/OS J2EE server.

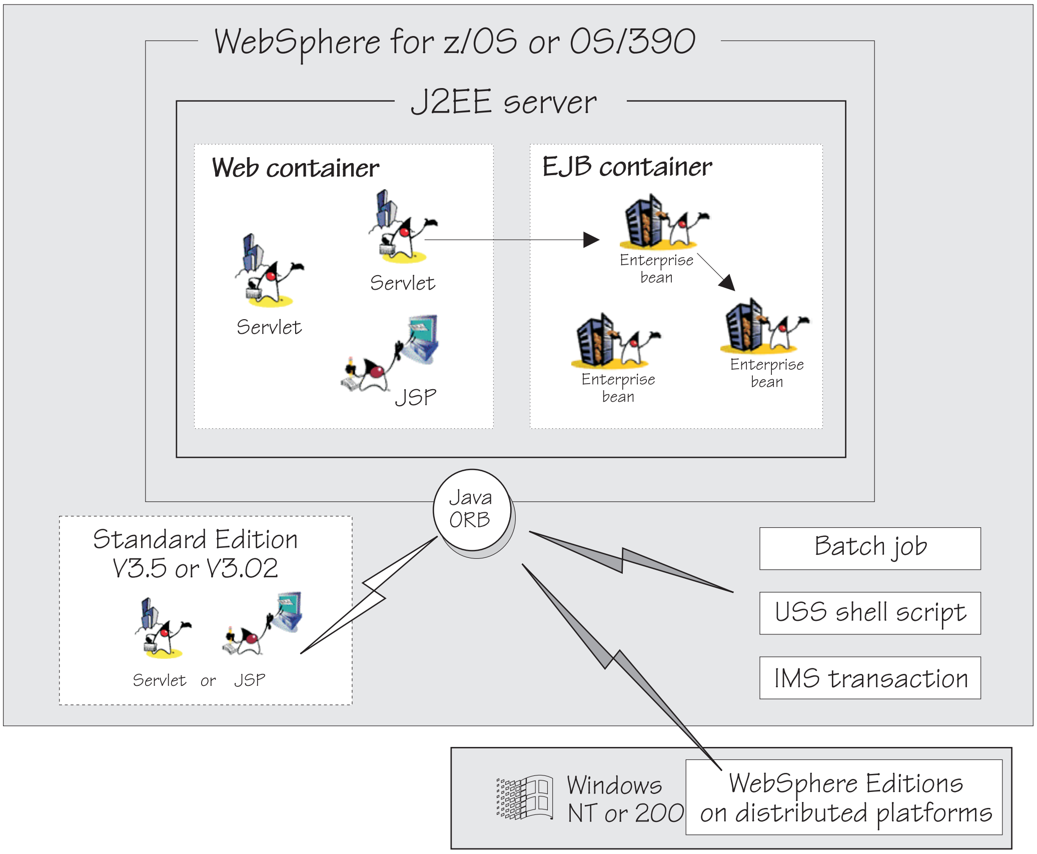

Figure 2. Potential clients of application components installed in the J2EE server

Figure 3 on page 6 illustrates the two possible configurations through which you can enable Web serving on z/OS or OS/390: Directing requests from the HTTP server to either:

1. Web applications in WebSphere Application Server Standard Edition Versions 3.5 or 3.02, or

2. Web applications in a Web container in a WebSphere for z/OS J2EE server. This option uses a WebSphere Application Server plug-in to direct requests to WebSphere for z/OS.

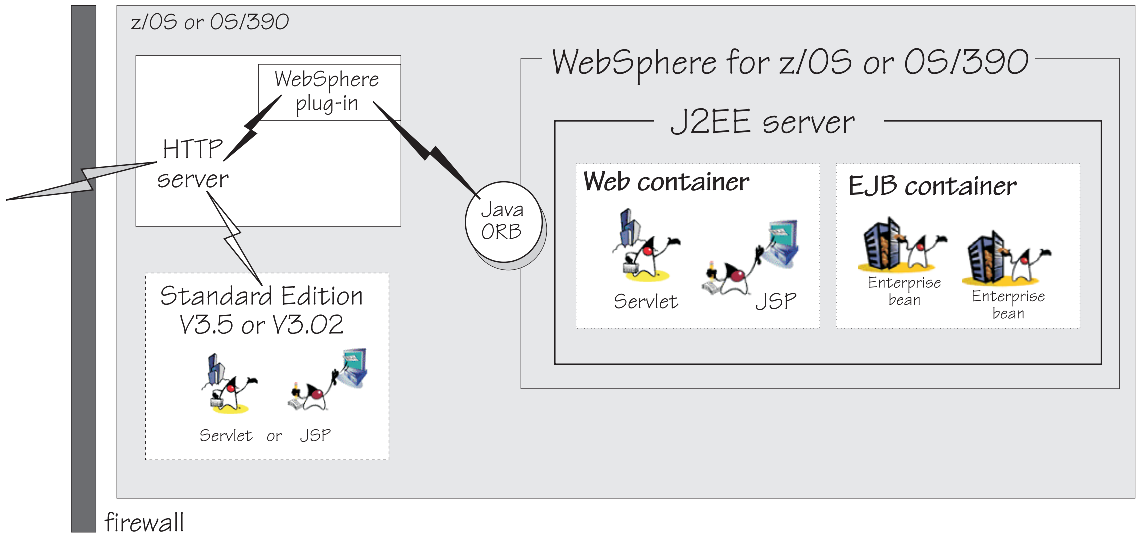

Figure 3. Possible configurations of Standard Edition for z/OS or OS/390 and the J2EE server

Because Enterprise beans and Web applications must conform to J2EE packaging standards, WebSphere for z/OS has introduced new 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 a new tool, the WebSphere for z/OS Application Assembly tool. See “Chapter 2. Overview of application tools” on page 7.
- 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 Application Server 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 11.

Chapter 2. Overview of application tools

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 4:

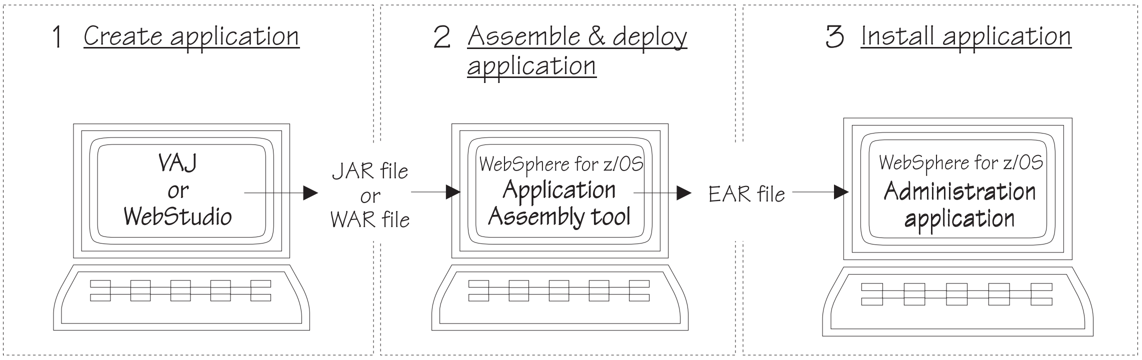
1 Create application
2 Assemble & deploy application
3 Install application
VAJ or WebStudio
JAR file or WAR file
WebSphere for z/OS Application Assembly tool
EAR file
WebSphere for z/OS Administration application

Figure 4. 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 VisualAge 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 5:
   - Enterprise beans, their classes and deployment descriptor packaged in Java archive (JAR) files
   - Web applications, consisting of servlets, their classes and descriptors, JSPs, and static files, such as HTML or GIFs, packaged in Web application archive (WAR) files

   Note that each JAR file may contain one or more Enterprise beans; similarly, each WAR file may contain multiple servlets or JSPs. These JAR and WAR files become the input for the next stage: assembly and deployment.

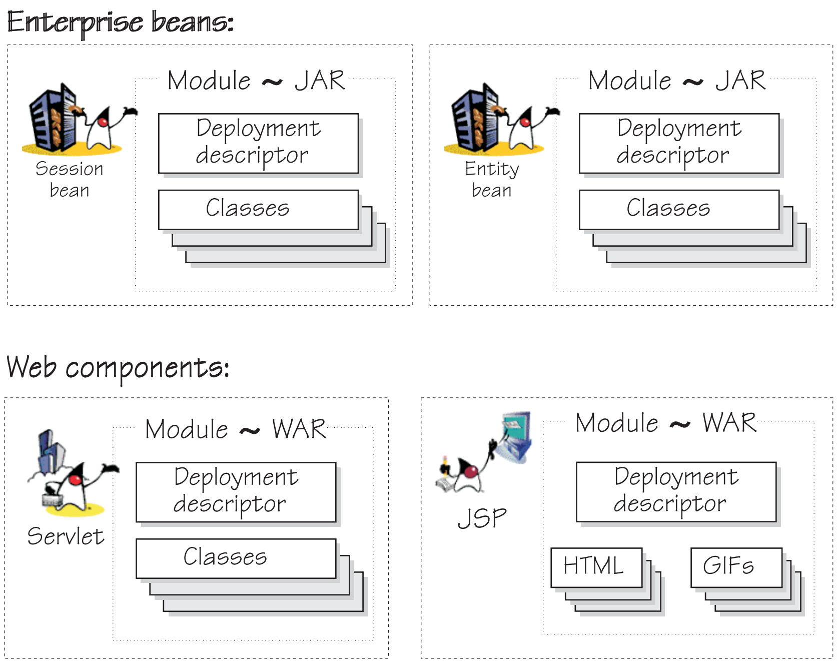

Figure 5. Supported J2EE application components

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.

   To assemble an application for installation on WebSphere for z/OS, you must use the WebSphere for z/OS Application Assembly tool, which:
   - Generates code for z/OS or OS/390, including remote interfaces, home interfaces, ties and stubs, keys, handles, finder helpers, and code related to persistence.
   - Converts 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

   To install an application on WebSphere for z/OS, you must use the WebSphere for z/OS Administration application. 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 JAR and EAR 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).

     Note: Only server-side `java:comp` look-ups are possible, because WebSphere for z/OS does not provide client container support. In other words, only Enterprise beans, servlets, and JSPs running in the J2EE server can use `java:comp` look-ups, as long as development tools used to create those components also support JNDI look-ups for `java:comp` names. Clients and applets must use explicit names for application components and homes.
   - 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 application server, which involves a combination of tasks that you complete by hand, and that you complete 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.

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 6 on page 12.

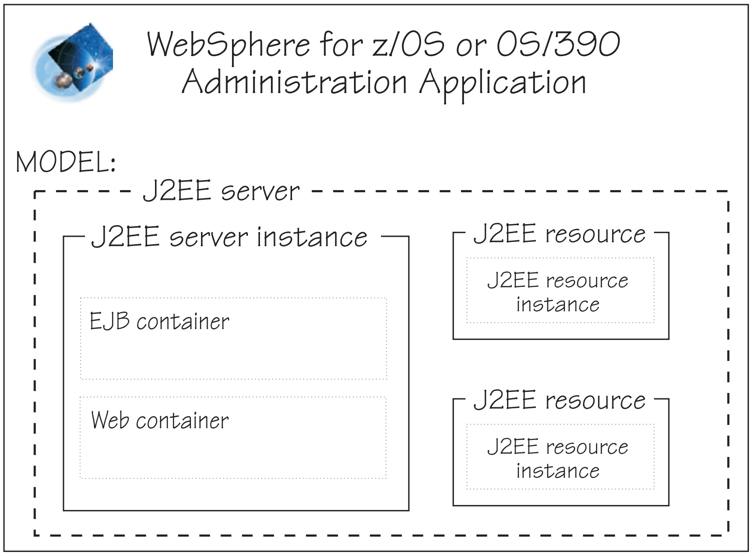

Figure 6. Model of a J2EE server

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 generic 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 instance in which your application will run. A server instance is an entity that represents, among other things, the Java virtual machine (JVM) in which your application components will run. 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.
- 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 JAR and EAR 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).

  Note: Only server-side `java:comp` look-ups are possible, because WebSphere for z/OS does not provide client container support. In other words, only Enterprise beans, servlets, and JSPs running in the J2EE server can use `java:comp` look-ups, as long as development tools used to create those components also support JNDI look-ups for `java:comp` names. Clients and applets must use explicit names for application components and homes.
- 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 7 on page 14 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.

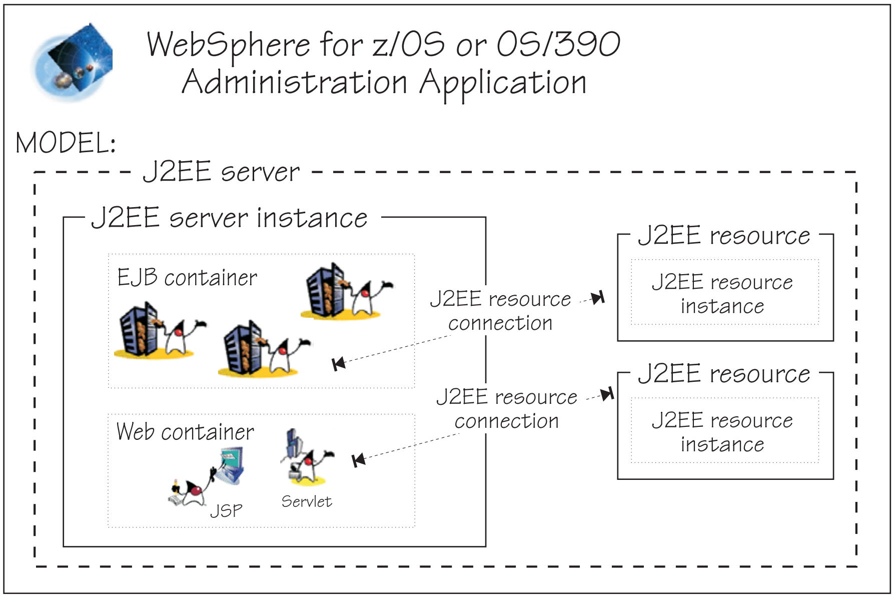

Figure 7. Installing a J2EE application in a J2EE server

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 8 on page 15 illustrates how the model elements correspond to active elements in the z/OS or OS/390 system.

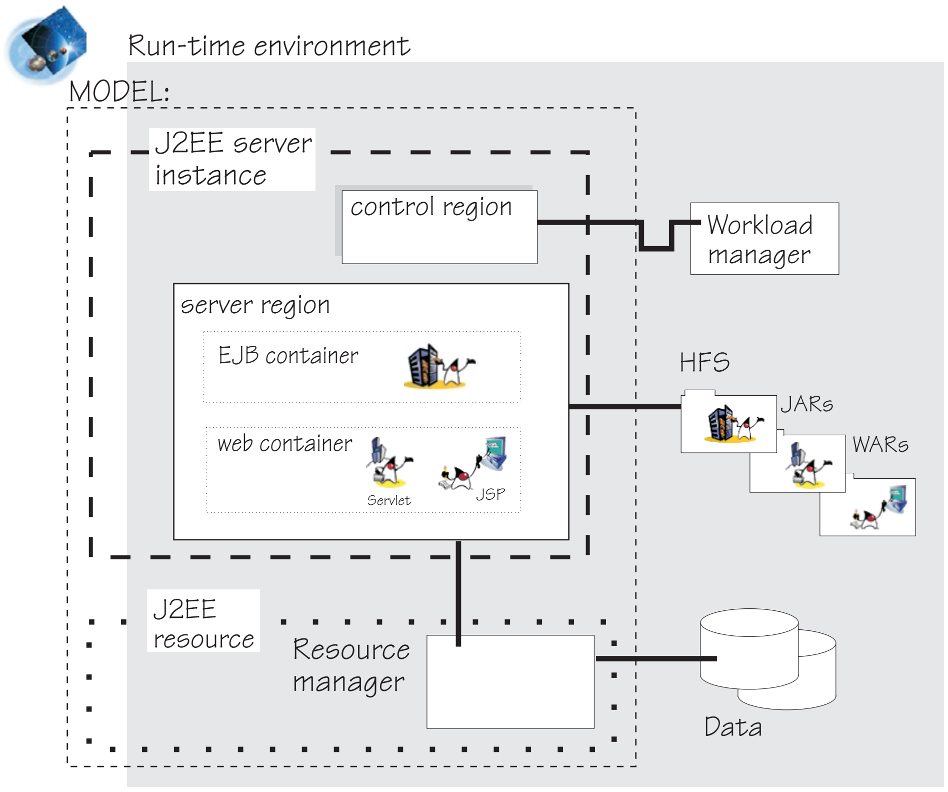

Figure 8. An active J2EE application server

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. 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 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.

When you activate a server model, WebSphere for z/OS connects the control region to WLM to manage client requests; and connects the server region to an actual subsystem, such as DB2, that manages data. As soon as a client request comes in, the J2EE server 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 the lifecycle begins.

Part 2. Creating, assembling and deploying J2EE server applications

Chapter 4. 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.

   Recommendation for developing EJB components: If you use the IBM EJB Development Environment feature of VisualAge for Java for developing and testing beans, servlets, and JSPs, you do not need to install the Java 2 Standard Edition (J2SE) Software Development Kit (SDK), or a WebSphere Application Server run-time. This feature of VisualAge for Java provides a code-generated test client and a WebSphere Application Server run-time that you can use to simulate the bean deployment environment on your workstation. The IBM EJB Development Environment 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.

   Recommendation for developing Web components: Use IBM WebSphere Studio to develop servlets or JSPs, and automatically package them into Web application archive (WAR) files. (If you use other tools, you might have to create the WAR files manually.) You can then use the WebSphere Application Server for Multiplatforms to test these Web components before assembling and installing them in a J2EE server.

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

J2EE application component	Software to use
Enterprise beans	**For development and testing:** • VisualAge for Java 3.5 with Patch 2, with the following features: – Data Access Beans 3.5 – IBM EJB Development Environment 3.5 – IBM Enterprise Extension Libraries 3.5 – IBM WebSphere Test Environment 3.5 – IBM Common Connector Framework 3.5 – IBM Enterprise Access Builder Library 3.5 – IBM Java Record Library 3.5 **Tip:** As an alternative to using VisualAge for Java, you may use non-IBM tools, such as JBuilder or Visual Cafe, for application development. Use the documentation for those products to determine hardware and software requirements. • IBM or Sun Microsystems Java 2 Standard Edition (J2SE) Software Development Kit (SDK)V1.3 • WebSphere Application Server Advanced Edition, V3.5, for testing application components. • (Optional) DB2 Universal Database Version 7.1, required only for testing beans that require the use of a persistent datastore.
	For assembly: The WebSphere for z/OS Application Assembly tool
	For installation in a J2EE server: TheWebSphere for z/OS Administration application
Servlets and JavaServer Pages (JSPs)	**For development and testing:** • WebSphere Studio 3.5.2 **Tip:** When you start WebSphere Studio, that tool checks to see that both VisualAge for Java and WebSphere Application Server Advanced Edition are installed on your workstation. • IBM or Sun Microsystems Java 2 Standard Edition (J2SE) Software Development Kit (SDK) V1.3
	For assembly: The WebSphere for z/OS Application Assembly tool
	For installation in a J2EE server: TheWebSphere for z/OS Administration application

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

Table 3. References for installation or migration information for application development software

Software:	**Source of installation or migration information:**
• VisualAge for Java • WebSphere Studio • WebSphere Application Server Advanced Edition for distributed platforms	For hardware requirements and installation instructions, 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 Application Server V3.5 (all editions for distributed platforms), for a complete listing of hardware and software. • The **InfoCenter** for the WebSphere Application Server Version 3.5 Standard and Advanced Editions 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.
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. Install or upgrade the appropriate assembly and deployment software on your workstation. For installation or migration instructions, see the following references:

Table 4. 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 33.
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 for z/OS and OS/390: Installation and Customization*, GA22-7834.

---

4. 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, these variables might be set automatically; refer to the installation documentation for each product for further details.

---

5. 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.

---

6. **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.

---

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. From the Command Prompt, 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.

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 may set 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 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 5. 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 4. Setting up the application development environment” on page 19, you are ready to create J2EE application components. Because documentation for developing J2EE application components is available through other WebSphere Application Server, Sun Microsystems, and third-party documents, the following topic lists 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 8. Creating and running J2EE application clients” on page 73.

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 27.

Perhaps the easiest approach to developing beans is to use the IBM VisualAge for Java EJB Development Environment, which enables you to code and test Enterprise beans that conform to the distributed component architecture defined in the EJB 1.0 specification. This feature of VisualAge for Java also provides a code-generated test client and a WebSphere Application Server run-time that you can use to simulate the bean deployment environment. The IBM EJB Development Environment enables developers to fully test entity and session beans, including JNDI lookups, remote method calls, and method calls on the home interface.

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 one of 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 VisualAge for Java IBM EJB Development Environment, follow these steps after the IBM home page loads into your browser:
    1. Click on the following links: `Products → Software → WebSphere → Products → VisualAge for Java`
    2. Click on `Library` in the left-hand frame, then scroll down to click on `Product Documentation`
    3. Click on the link for `IBM VisualAge for Java 3.5 PDF Documentation`
    4. Scroll down to the category `Developing EJB components` and click on `ejb.pdf`

Table 5. Checklist for developing Enterprise beans

✓	**Task**
☐	Add a new project and package for the beans you are creating **Note:** A project is a concept used in VisualAge for Java only.
☐	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, DB2 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 DB2 tables
☐	Add properties, including: • The JNDI name for the BeanHome • Transaction attribute
☐	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 "Packaging beans in JAR files" on page 28

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:

- Container-managed persistence (CMP) is supported only for Enterprise beans created using VisualAge for Java.
- 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.

Guidelines:

- You may code your beans to use the JDBC application programming interface (API) to access DB2 data. For additional information, see:
    – *DB2 Application Programming and SQL Guide*, SC26-8958 for instructions on using the DB2 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 DB2 to store persistent data, you need to use the type 2 JDBC driver that ships with DB2 for 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 DB2. (Note, however, that performance is better with only one connection.)
    – Assign a different outcome for each DB2 connection, when the bean does not run under a global transaction.

Packaging beans in JAR files

If you use VisualAge for Java to develop and test Enterprise beans, you package the beans by selecting the `Export → Deployed Jar` option. During this process, you select the beans, classes, and resources to be included in the JAR file.

Recommendations: To limit the JAR file contents to only those application parts that you need on z/OS or OS/390, which should make future maintenance of the application easier, do the following:

- Review the details for classes that VisualAge for Java will export. Classes with a checkmark to the left will be exported unless you click on that mark to deselect the class. For example, if this bean uses DB2 as a persistent datastore, VisualAge for Java automatically includes the DB2 JDBC classes you imported to create the bean. Because these classes reside on z/OS or OS/390 already, you do not have to include them in your bean JAR file.

- Consider exporting JAR files without using the VisualAge for Java `Select referenced types and resources` option. When you package your application (as described in "Chapter 6. Assembling a J2EE application" on page 33), the resulting EAR file will contain the necessary classes for the J2EE server.

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.)
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 35 for information on how to perform this packaging.) An EAR file is basically a JAR file with a specific directory structure and format and has an extension of .ear. It includes a 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.)

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.

Packaging Web components in WAR files

A tool, such as the IBM WebSphere Studio product, that is used to create JAR files can be used to create a WAR file as long as the resulting WAR file:

- Includes a web.xml file in the `WEB-INF` directory.
- Any JAR files being included reside in the `WEB-INF`/lib directory
- The files and JSPs that are being included are rooted at the top of the jar.

The following is an example of a war file that includes a servlet, servlet class files, a JSP and static files:

```
/mywebApplication
   index.html
   other.html
   CatalogDisplay.jsp
   /gifs
      logo.gif
   /WEB-INF
      web.xml
      /classes
         /com
            /mycorp
               /CatalogServlet.class
      /lib
         myItems.jar
```

The web.xml file contained in this WAR file might look like the following:

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc... ...dtd"><
web-app><
display-name>A Simple Application</display-name><
context-param><
param-name>Webmaster</param-name><
param-value>webmaster@mycorp.com</param-value><
/context-param>

<servlet><
servlet-name> catalog</servlet-name><
```

```
servlet-class>com.mycorp.CatalogServlet</servlet-class><
init-param><
param-name>catalog</param-name><
param-value>Spring</param-value><
/init-param><
/servlet><
servlet mapping><
servlet-name>catalog</servlet-name><
url-pattern>/catalog</url-pattern><
/servlet mapping>
```

Chapter 6. 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 basically 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 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. The Application Assembly tool generates 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.

In addition to packaging new applications, you may use the Application Assembly tool to change the contents of existing J2EE applications and their associated EAR files.

The following table shows the subtasks and associated procedures for using the Application Assembly tool to assemble a J2EE application:

Subtask	**Associated procedure (See . . .)**
Install the Application Assembly tool on your workstation	“Steps for installing the Application Assembly tool”
Assemble a new J2EE application	“Steps for assembling a new J2EE application” on page 35

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/`). From that site, click

| on `Download` in the left frame, and scroll to the link for `WebSphere
| Application Server V4.0 for z/OS and OS/390 downloads` 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 390 Application Assembly` under `C:\Program Files\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 4. Setting up the application development environment" on page 19, and
- "Chapter 5. Creating new application components to be installed in a J2EE server" on page 25.

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 33.
- 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.

   ---
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

   ---

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 the following message if it detects errors in any deployment descriptors:

   ```
   BBO94030E Error(s) detected in application_name deployment descriptors.
   See message log for details.
   ```

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 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. These component properties correspond to the appropriate Sun Microsystems 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, specify role names in the deployment descriptor for either individual components or for the J2EE application. Application-level roles override component-level roles.

    Rules:
    - A role name cannot contain blanks, and cannot exceed 246 characters. Role names, however, may be in mixed case.
    - Role names must match profiles specified in the EJBROLE or GEJBROLE class, which a security administrator defines for your installation. (Instructions for defining these classes appear in "Steps for completing manual OS/390 tasks" on page 44.)
    - For each security role that you define for an individual component, make sure you select the component methods to which each role has access. Use the method permissions property form to do so.
    - If beans or servlets use the `isCallerInRole` or `getCallerPrincipal` methods, these security role reference names must be linked to the security roles you have defined to match profiles specified in the EJBROLE or GEJBROLE class. Use the Security tab for an individual component to define these links. Under the Security tab, fill in the following information related to security role references:

Role name	Description	Link
The string used on the `isCallerInRole` or `getCallerPrincipal` method. **Note:** If you are working with an Enterprise bean written to the 1.1 specification level, these strings will appear automatically.	An optional description that explains the level of authority required for this security role	The security role name that you have defined to match a specific profile in the EJBROLE or GEJBROLE class

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 was exported to the EAR file.

Chapter 7. Creating a J2EE server run-time environment

Now that you have created an EAR file to use to install your J2EE application, you need to create theWebSphere for z/OS J2EE server in which your application will run. The J2EE server, or run-time environment, consists of the following elements:

- A generic server that represents the application environment. A server is an entity that is responsible for a certain type of work.
- A 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 and specific 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 OS/390 tasks" on page 44 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 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 OS/390 tasks	"Steps for completing manual OS/390 tasks" on page 44

Subtask	Associated procedure (See . . .)
Creating JCL procedures	“Steps for creating JCL procedures for the control and server regions” on page 46
Setting JVM properties	“Steps for setting properties for the JVM” on page 47
Enabling the J2EE server to host Web applications (optional)	“Enabling J2EE server support for Web applications (optional)” on page 47
Defining and activating the J2EE server through the Administration application	“Defining the server configuration” on page 62

Steps for completing manual 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 for z/OS product itself, or setting up WebSphere for z/OS 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 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`

Field in IWMARIN0 dialog:	Value to use:
Application environment name	`J2SERV`
Subsystem type	`CB`
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*, SC26-8957 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 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 SecureWay 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'`

Steps for setting properties for the JVM

Use the following procedure 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 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 150, 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 for z/OS 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 for z/OS 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 for z/OS cannot find or use the property file you provide, it continues the process of activating the server, using default JVM property values.

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 62. If your application contains
| servlets or JSPs, however, you must complete some additional tasks to set up
| the J2EE server configuration.

The following table shows the subtasks and associated procedures for enabling support for Web applications installed in a WebSphere for z/OS J2EE server:

Subtask:	Associated procedure (See ...)
Configuring a Web container for servlets or JSPs to be installed in the J2EE server	1. "Setting up a Web container in a J2EE server" 2. "Exposing Web applications to HTTP clients" on page 52
Setting up an HTTP server to establish communication between the J2EE server and a Web browser	"Configuring HTTP session support" on page 54

Setting up a Web container in a J2EE server

Each WebSphere for z/OS J2EE server contains at least one Web container and one EJB container. The Web container is able to manage Web applications in accordance with the guidelines described in the Java™ Servlet Specification V2.2. The Web container also provides support for managing JavaServer Pages that are compliant with the Javasoft JavaServer Pages V1.1 Specification.

The J2EE server within which a Web container resides, provides additional services for the applications deployed in that Web container, such as security and resource management. It also enables a Web application to access external resources such as relational databases through JDBC, and Enterprise Java Beans, over RMI.

HTTP work management in a sysplex is not changed with the introduction of WebSphere for z/OS. OS/390 Web servers still act as the communication endpoint for HTTP requests that are inbound into the sysplex. Installations are not required to change their existing HTTP Server and network topologies. Existing outboard routing and workload distribution mechanisms, such as intelligent routers and network dispatchers, remain valid in this environment.

Any OS/390 HTTP Server that is going to be used to route work to a J2EE server in the sysplex must be configured to work with the plugin routine that is provided with the WebSphere for z/OS product. The plugin routine will work with the OS/390 Workload Manager to determine the best J2EE server in which to service each HTTP request. "Exposing Web applications to HTTP clients" on page 52 describes the changes that need to be made to an HTTP Server's httpd.conf file and httpd.envarrs file to configure it to work with the plugin routine.

Customizing the Web container in a J2EE server

A Web container is created as part of the J2EE server set up process. Its 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. When the Web container is initially configured, at least one virtual host (the default virtual host that is provided with the product) is associated with it.
- 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. “Configuring HTTP session support” on page 54 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.

The default webcontainer.conf file that is shipped with the product is located in the *applicationserver_root*/bin directory.

The following property is used to associate a webcontainer.conf file with a J2EE server. You must add this property to the target J2EE server’ JVM properties file to enable the J2EE server to recognize your customized webcontainer.conf file:

```
com.ibm.ws390.wc.config.filename=/your_root/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 configuation files associated with this J2EE server.

After editing the webcontainer.conf file, you must refresh the J2EE server to activate the changes you made.

Configuring a virtual host: 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 in accordance with the hosts for which it is servicing requests.

Properties in the webcontainer.conf file of the form **host.**<*virtual-hostname*>**.**<*property*>**=**<*value*> are used to define a virtual host. These properties indicate the name by which this host is known within the WebSphere for z/OS administrative domain (*virtual-hostname*).

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.

The following properties are used to configure a virtual host:

- **host.**<*virtual-hostname*>**.alias**=<*hostname*>. This property specifies a hostname alias to be associated with this virtual host name. It provides a binding between the host names understood by 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**. This property is the fully qualified name of a file containing definitions for MIME types that describe the content that can be included in HTTP responses served from this virtual host. The Web container contains a default MIME type file containing standard MIME type definitions. The name of this default file is contained in the default webcontainer.conf file provided with the product.
- **host.**<*virtual-hostname*>**.contextroots**. This property is used to bind installed Web applications into a specific virtual host. The context root specified 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 contextroots 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 153 for more a more detailed description of this property.

A virtual host can have more than one alias. The alias definition may contain both a host name and a port number. When a client requests a Web application, servlet, or related resource, WebSphere for z/OS compares the hostname and port in the request with the list of configured DNS aliases. If a match is not found, WebSphere for z/OS reports an error that is returned to the browser. The following example illustrates the properties you might include in the webcontainer.conf file to configure the virtual host MyHost with DNS aliases of www.mycompany.com and www.MyOtherCompany.com:

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

See "Appendix B. Default webcontainer.conf file" on page 153 for a complete description of the webcontainer.conf properties that are applicable to defining a virtual host.

Installing Web applications into a J2EE server

A Web application exists within a single WebSphere for z/OS instance. It can be replicated, if it is marked distributable. It uses servlet context to obtain references to other local objects and to share data with other applications.

A Web application consists of various Web components, such as:

- Servlets
- JavaServer Pages (JSPs)
- Utility classes
- Static documents

The role each Web component plays in a Web application is defined in the Java Servlet Specification V2.2, which is available at the following URL:

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

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 Archive (WAR) file.

   Note: A tool, such as the IBM WebSphere Studio product, that is used to create JAR files can be used to create a WAR file. However, after creating the WAR file, check the web.xml file that the tool will also create, to make sure that all of the <servlet>XML tags are grouped together; not intermixed with the <servlet-mapping> tags. Some tools intermix the <servlet>tags with <servlet-mapping> tags, which can create processing errors. If the tags are intermixed, edit this file and group all of the <servlet> tags together and group all of the <servlet-mapping> tags together.
2. This WAR file must then be packaged as part of an Enterprise Archive (EAR) file. An EAR file is basically a JAR file with a specific directory structure and format and has an extension of .ear. It includes a application.xml file which contains the descriptive meta information which ties together all of the WAR and/or EJB JAR files packaged in the EAR file.

   Use the Application Assembly Tool for z/OS and OS/390, that is provided with WebSphere for z/OS, to create EAR files. This tool requires as input, the WAR files and/or EJB JAR files you want included in the EAR file.

The Systems Management EUI provided with WebSphere for z/OS is used to install a Web applications into the Web Container. The application must be in the form of an Enterprise Application Archive file (.ear file). Systems Management EUI takes the .ear file, resolves references and installs the Enterprise application into the Web container.

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.)

It is the responsibility of the Application Assembler to create the enterprise application package (EAR file plus application.xml file) and ensure that all component references can be resolved.

Exposing Web applications to HTTP clients

A Web application that is installed in a J2EE server needs to be made accessible to HTTP clients, such as Web browsers. Therefore, WebSphere for z/OS requires that at least one OS/390 HTTP Server is defined within the sysplex. The plugin routine provided with WebSphere for z/OS can then enable the HTTP server to find Web applications that are installed in J2EE servers within the sysplex. When the plugin receives an HTTP request, it routes the request to the appropriate J2EE server for processing.

Before an HTTP server can communicate with a J2EE server, you must:

1. Add the following Web server directives to the httpd.conf configuration file of any Web server that will be communicating with WebSphere for z/OS to provide the Web server with the entry point to the WebSphere for z/OS plugin's initialization, request processing, and exit routines. These routines exist as entry points init_exit, service_exit, and term_exit, respectively, within the was400plugin.so DLL. The was400plugin.so DLL is found within the *applicationserver_root*/**WebServerPlugIn/bin** directory.

   ```
   ServerInit applicationserver_root/WebServerPlugIn/bin/
      was400plugin.so:init_exit  applicationserver_root,was.conf_name
   ServerTerm applicationserver_root/WebServerPlugIn/bin/was400plugin.so:term_exit
   Service /webapp/* applicationserver_root/WebServerPlugIn/bin/
      was400plugin.so:service_exit
   ```

   applicationserver_root is the fully qualified name of the mounted install-image of an individual execution system. The default value is **/usr/lpp/WebSphere**

   was.conf_name is the fully qualified name of a V3.5 was.conf file. This parameter is optional and is only required if you want to continue using your V3.5 Application Server along with WebSphere for z/OS V4.0 . See "Migrating from version 3.5" on page 161 for a description of the changes you need to make to your V3.5 Application Server was.conf file in order to continue using V3.5 in a V4.0 environment.

Notes:

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

b. 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.

2. Make sure that the JAVA_HOME environment variable contained in the hosting Web server's httpd.envvars file (as well as any other environment variable, such as PATH or LIBPATH) points to the exact location where the required level of the Software Development Kit (SDK) is installed on your system
3. Append the WebSphere for z/OS plugin's message catalog directory to the existing NLSPATH specified in the HTTP server's envvars file. For example, if NLSPATH was set as:

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

and the WebSphere for z/OS plugin is installed in **/usr/lpp/WebSphere**, change the NLSPATH to:

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

4. Start the HTTP server

Once the HTTP server is started, a client can use a Web browser to initiate a transaction with a Web application. This transaction is communicated via the HTTP server to the appropriate Web container residing in the J2EE server. If the OS/390 system is a sysplex with multiple J2EE servers, WebSphere for z/OS determines which J2EE server contains the correct Web container.

Multiple requests can be initiated concurrently to different replicated J2EE servers. WebSphere for z/OS will serialize these requests within a single session across containers.

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

Configuring HTTP session support

A session is a series of requests originating from the same user, at the same browser. Using WebSphere for z/OS's implementation of the Java Servlet API session framework, your Web container can maintain state information about sessions.

WebSphere for z/OS provides facilities we group under the heading Session Manager that support the javax.servlet.http.HttpSession interface described in the Servlet API specification. A session object can be implemented in a variety of ways, each of which provides different levels of performance, failover, and clustering. In all cases, WebSphere for z/OS 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.

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 to execute concurrently in multiple Application Server instances. If two requests that are part of the same session arrive at two different Application Server instances, WebSphere for z/OS will serialize the dispatch of these requests among the Application Servers.

WebSphere for z/OS allows multiple requests in the same session to execute concurrently within the same Application Server instance. It is the responsibility of the Web application components (servlets, JSPs, etc.) to serialize their access to the HTTP Session object within the same Application Server. WebSphere for z/OS maintains the responsibility of providing the serialization among Application Server instances.

WebSphere for z/OS makes use of a DB2 database as the mechanism for serializing access to and sharing HTTP Session State data. It uses the same HTTP Session database format as the Versions 3.5 and 3.02. Therefore, the administrator is not required to create new databases for Version 4.0. Instead, he can allow Versions 4.0, 3.5 and 3.02 to concurrently utilize the same database in their processing.

Maintaining HTTP Session State data in-memory is still supported in WebSphere for z/OS Version 4.0. When maintaining HTTP Session data in-memory, it is unable to be shared across multiple instances of the Web application that exist concurrently in multiple Application Server regions. If HTTP Session data is configured to be in-memory, it is necessary for the Web application that accesses that HTTP Session data to be placed in a J2EE server which has been configured to have only one control region defined in the sysplex and only one server region defined for that control region. The ability to constrain the number of runtime instances of a J2EE server is controlled by OS/390 Workload Manager policy.

Configuring session tracking

Each plugin routine contains a single Session Manager. The Session Manager supports the javax.servlet.http.HttpSession interface described in the Java Servlet API 2.1 specification. When configuring the Session Manager, the WebSphere administrator can specify:

- Whether to enable sessions.
- How to convey session IDs to servlets (cookies or URL rewriting).
- Whether to save session data in a DB2 database during execution (persistent sessions)
- Whether to add session IDs to URLs in transition from HTTP to HTTPS and back (protocol switch rewriting)

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

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

Note: This example illustrates a minimal set of options. The full set of session properties, including detailed descriptions, are provided in the default webcontainer.conf file, a copy of which is provide in “Appendix B. Default webcontainer.conf file” on page 153.

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 getUserName method on the HTTP Request object as the user name associated with a session. If the getUserName 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 to be used with session tracking, the following changes might need to be made to properties in the webcontainer.conf file:

- Set the **session.cookies.enable** property to **true** to enable cookies.
- Specify the name of the cookie on the **session.cookie.name**property.
- 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.)
- Set the **session.cookie.path** to a string that specifies to which paths on the HTTP server cookies will be sent. This change is only needed if you want to restrict to which servlets, JHTML files, and HTML files cookies will be sent.
- Set the **session.cookie.domain** property to a specific name if you want to limit the domain for which a cookie is valid.

- Add a **session.cookie.comment** property if you want to include a comment about the cookie.
- Set the **session.cookie.secure** property to **true** if you want to restrict the exchange of cookies to only HTTPS sessions.

The Session Tracker 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 Tracker uses it to find the HttpSession associated with this user.

Using URL rewriting

To use URL rewriting, you must set the **session.urlrewriting.enable** and **session.protocolswitchrewriting.enable** properties in the webcontainer.conf file to **true**. These settings:

- Enable URL rewriting in the Session Manager, using a servlet or a JSP as an entry point. This entry point is not dependent on sessions for its processing; rather, it contains encoded HREFs to servlets in the application that are dependent on sessions.
- Enable the session ID to be added to a URL when the URL requires a switch from HTTP to HTTPS, or HTTPS to HTTP.

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

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

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

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

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

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

then this code must be replaced with:

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

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

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

then this code must be replaced with:

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

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

Session clustering

To support propagating events across z/OS or OS/390 nodes in a session cluster, WebSphere for z/OS uses a database to track and manage sessions in the common pool of sessions across all z/OS or OS/390 cluster nodes. With the use of a database as well as the general architectural changes implemented in this version of WebSphere for z/OS, WebSphere for z/OS no longer maintains the notion of a session cluster client and a session cluster server. In a clustered environment, the session may be accessed on any one of the virtual hosts in a cluster; which one is actually accessed will be transparent to the end user.

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

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

WebSphere for z/OS uses a DB2 database as the sharing mechanism among WebSphere for z/OS instances. Any V3.5 Application Server that is executing on a z/OS or OS/390 image and is able to access the central database is able to participate in the session cluster.

To configure a session cluster, you must:

- Have your DB2 Administrator create a database table for use within the session cluster. (For more information about creating DB2 tables see the DB2 Administration Guide for the version of DB2 you will be 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 TABLESPACE <tablespace_name>
           IN <database_name>
           USING STOGROUP <group_name>
                PRIQTY 52
                SECQTY 2
                ERASE NO
           LOCKSIZE ROW
           BUFFERPOOL BP32K
           CLOSE YES;
  ```

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

  ```
  CREATE TABLE TABLEOWNER.<table_name>
           (ID              VARCHAR(24)     NOT NULL,
           PROPID           VARCHAR(24)     NOT NULL,
           APPNAME          VARCHAR(32),
           LISTENERCNT      SMALLINT,
           EXPIRES          TIMESTAMP,
           LASTACCESS       TIMESTAMP,
           CREATIONTIME     TIMESTAMP,
           MAXINACTIVETIME  INTEGER,
  ```

```
         USERNAME         VARCHAR(256),
         SMALL            VARCHAR(3595)   FOR BIT DATA,
         MEDIUM           LONG VARCHAR    FOR BIT DATA
         )
IN DATABASE.<database_name>;
```

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

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

Note: 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 Application Server that is configured to use persistent sessions should be granted both read and update access to the subject database table.

- Make sure that the following property settings are specified in the webcontainer.conf file to enable session persistence and to inform the Session Manager of the location of its entities:

```
session.enable=true
session.invalidationtime=<milliseconds>
session.cookies.enable=true
session.dbenable=true
session.dbjdbcpoolname=<session-jdbc-poolname>
session.datasourcename=<datasourcename>
session.dbtablename=<database-tablename>
```

<milliseconds> is the amount of time in, milliseconds, that a session is allowed to go unused before it is considered invalid.

<session-jdbc-poolname> is the name of the JDBC database connection pool that will be used for session support whenever the **session.dbenable** property is set to true.

<datasourcename> is the name of the datasource for this JDBC database connection pool.

<database-tablename> is the name of the database table that is to be used by the session services.

You can also change the value on the **session.tableoverflowenable** to false if you want to limit the number of session objects maintained by the WebSphere for z/OS plugin to the number of session objects specified on the **session.tablesize** property. (The default value for the **session.tablesize** property is 1000 session objects.)

Session clustering considerations: You should be aware of the following caveats regarding how session management works within a clustered HTTP server environment:

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

In-memory session pools

You can specify the number of in-memory sessions that are to be maintained. Once this number is surpassed, these functions are bypassed. General memory requirements for your hardware system, as well as your site's usage characteristics, will determine the optimum value for this number. Also, with larger numbers, you may need to increase the heap sizes of the Java processes for WebSphere for z/OS instances.

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

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

Defining the server configuration

Use the WebSphere for z/OS 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 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 123 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 for z/OS Administration application:

Subtask:	Associated procedure (See ...)
Starting the Administration application	"Steps for starting the Administration application" on page 63
Starting a new conversation	"Steps for starting a conversation" on page 63
Adding the server	"Steps for adding the J2SERV server" on page 64
Adding the server instance	"Steps for adding the J2SERV1 server instance" on page 66
Adding the J2EE resource	"Steps for adding a J2EE resource" on page 66

Subtask:	Associated procedure (See ...)
Adding the J2EE resource instance	“Steps for adding the J2EE resource instance” on page 67
Installing the J2EE application	“Steps for installing a J2EE application” on page 67
Validating the new conversation	“Steps for validating the new conversation model” on page 70
Committing the conversation	“Steps for committing the conversation” on page 70
Marking manual tasks as completed	“Steps for marking z/OS or OS/390 tasks as completed” on page 70
Activating the new conversation	“Steps for activating the server configuration” on page 71

Steps for starting the Administration application

Before you begin: You need to know the naming server IP name and port number for the machine running WebSphere for z/OS. The naming server IP name is set either in your domain name server (DNS) or workstation `HOSTS` file; the default port number is 900.

Perform these steps to start the Administration application:

1. On your workstation, click on Start, then Programs, then 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 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

Environment variable list	Provide values for the following key environment variables for the application server.
	Note: Make sure you set all **required** environment variables for the run-time environment. See "Appendix A. Environment and JVM properties files" on page 123 for a complete list of application server run-time variables and their values. You may also browse the `current.env` file to look up current values. Then cut-and-paste the existing value into the panel and add to it, if necessary. Use quick keys for cut/copy and paste ([ctrl]+c for COPY, [ctrl]+x for CUT, [ctrl]+v for PASTE). These functions are not available from a pop-up menu in the tables for the environment variables.
	• LIBPATH. The LIBPATH variable specifies the DLL search paths for Java and JDBC in the hierarchical file system (HFS). Specify system, WebSphere for z/OS, 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 for 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`
	
	Note: After activation of this conversation, WebSphere for z/OS automatically prepends the following files to the J2EE server CLASSPATH for you:
	– `ws390srt.jar`
	– `waswebc.jar`
	– `xerces.jar`

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
   - Database name: supply the DB2 for OS/390 location name

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.

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 63.

- 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. Then complete the following steps, as necessary, for each bean:
   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.
   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 for 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 for z/OS 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. 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 "Exposing Web applications to HTTP clients" on page 52 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 OS/390 tasks" on page 44.

---

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.

Chapter 8. Creating and running J2EE application clients

Once you have installed application components in a J2EE server, you are ready to create J2EE application clients that use those components. Figure 2 on page 5 illustrates the most likely types of clients for components that run in a J2EE server. The information in this chapter primarily applies to z/OS or OS/390 clients, but cites sources of details for developing and running non-z/OS or non-OS/390 clients.

The following table lists types of client applications, and where to find information about creating and running them. For all types of clients, make sure you read the information in "Security considerations for J2EE applications" on page 79.

For this type of client:	See the following information sources:
Clients running in WebSphere Application Server Standard Edition on z/OS or OS/390	"WebSphere Application Server Standard Edition for z/OS or OS/390 clients"
Clients running on z/OS or OS/390	"Native z/OS or OS/390 Java clients" on page 76
Clients running on WebSphere Application Server on distributed platforms	"WebSphere Application Server Advanced Edition and Standard Edition clients on non-z/OS and non-OS/390 platforms" on page 77

WebSphere Application Server Standard Edition for z/OS or OS/390 clients

Figure 3 on page 6 depicts possible configurations for WebSphere Application Server Standard Edition Web applications. The following information applies only to the case in which Web applications run in the Standard Edition environment and drive Enterprise beans that run in a WebSphere for z/OS J2EE server.

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. **Before you begin:** You must have the WebSphere Application Server Standard Edition Version 3.5 for 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 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

Perform the following steps to create and run J2EE application clients:

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 4. Setting up the application development environment" on page 19.

   Recommendations:

   - Use VisualAge for Java or WebSphere Studio to develop and test servlets and JSPs. The VisualAge for Java WebSphere Test Environment feature enables 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 key
   | `java.naming.factory.initial` to
   | `com.ibm.websphere.naming.WsnInitialContextFactory`.

   Recommendations:

   - Instead of hardcoding property values in the client application itself, set the properties through the `was.conf` file for the WebSphere Application Server. See Step 6 on page 75.
   - 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 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 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/was302plugin.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
   ```

   where *path* is the directory in which the jar file resides.
   | `/usr/lpp/WebSphere/lib/` is the default path.

   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 J2EE 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 applications on z/OS or OS/390 from a Java client running on z/OS or OS/390. *WebSphere Application Server V4.0 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 because WebSphere for z/OS does not provide a client container. 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:

    ```
    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.
- **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 and consult with your security administrator.

WebSphere Application Server Advanced Edition and Standard Edition clients on non-z/OS and non-OS/390 platforms

When your installation's system programmers install and customize WebSphere for z/OS, they run a sample J2EE application (which is shipped as part of the product) to verify that the installation was successful. You can use the same application to see how to drive J2EE applications on z/OS or OS/390 from a remote platform, such as Windows NT.

To drive the WebSphere for z/OS installation verification program (IVP) remotely, you may use the WebSphere for z/OS Java Client for Windows, which is available through the WebSphere Application Server Web site. The WebSphere for z/OS Java Client for Windows package contains the Java client Policy IVP, which is the same IVP used on z/OS or OS/390 as part of the WebSphere for z/OS installation process. The WebSphere for z/OS Java Client for Windows also contains a `README` file with further instructions for running the Policy IVP client. Use these instructions as a model for setting up your own clients on Windows to drive Enterprise beans installed in a WebSphere for z/OS J2EE server.

Depending on the software installed on your workstation, you might not need to download the WebSphere for z/OS Java Client for Windows to run Java clients that use J2EE application components installed in WebSphere for z/OS. Use the following chart to determine when the Java Client for Windows download package is required:

If your workstation configuration:	Then:
Includes WebSphere Application Server Advanced Edition V4.0	You **do not** need to download the WebSphere for z/OS Java Client.
Includes WebSphere Application Server Advanced Edition V3.5	You **must** download the WebSphere for z/OS Java Client to run clients that use J2EE application components installed in WebSphere for z/OS.
Does not include any version of WebSphere Application Server Advanced Edition	

To download the WebSphere for z/OS Java Client for Windows, go to the WebSphere Application Server Web site:

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

Then click on `Download` in the left frame, and scroll to the link for `WebSphere Application Server V4.0 for z/OS and OS/390 downloads` to access the WebSphere for z/OS Java Client for Windows.

Rules:

- Clients must include the following on their CLASSPATH: 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:

  ```
  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.

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 and consult with your security administrator.

For further information about designing and coding J2EE client applications to run on platforms other than z/OS and OS/390, see the **InfoCenter** information for the WebSphere Application Server product you are using. The **InfoCenter** is available at `http://www.ibm.com/software/webservers/appserv/`

Security considerations for J2EE applications

The security profiles and permissions for clients 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. If your installation does require the authentication of clients, however, you might need to work with your installation's security administrator to comply with security requirements.

Guidelines:

- For recommendations and instructions for setting up security for J2EE servers and J2EE application clients, see *WebSphere Application Server V4.0 for z/OS and OS/390: Installation and Customization*. The security topics in that book cover:
  - Client authentication
  - Client access to J2EE servers
  - Client access to objects in a server
  - Client access to DB2 (optional)
- If your client applications need permission to use Enterprise bean or servlet methods that are protected through 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 SecureWay 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.

Part 3. Working with J2EE applications in the run-time environment

Chapter 9. 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 85

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.
   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 resources 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 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 exactly the same 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 for z/OS and OS/390: System Management Scripting API*, SA22-7839.

Chapter 10. 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 JRas support to enable applications to issue messages and trace entries	"Logging messages and trace data for Java applications"

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 for z/OS and OS/390: Operations and Administration*, SA22-7835.

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 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 *Using the WebSphere JRas Message Logging and Trace Facility*, which describes the 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" • "Background on issuing trace requests for your application" on page 91
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 93
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 99
Viewing messages or trace data collected for your Java server application	• "Background on viewing messages and trace data" on page 102 • "Steps for using IPCS in batch mode to format application trace data" on page 103

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 93. 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 91

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 93 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 93. 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” on page 93

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 99 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 90 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 *Using the WebSphere JRas Message Logging and Trace Facility*, which describes the 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 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 99 .

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 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 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 6. 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 6. 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 123 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 17.

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 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". See *WebSphere Application Server V4.0 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 for z/OS and OS/390: Messages and Diagnosis*, GA22-7837.

Part 4. Migrating applications to the J2EE server

The following chapters contain information about migrating applications to the WebSphere for z/OS J2EE server environment. Because WebSphere for z/OS V4.0 requires compliance with the latest J2EE programming and packaging specifications, and requires the use of specific development, assembly, and installation tools, consider reading the following introductory material before using the instructions to migrate application components:

- "Chapter 2. Overview of application tools" on page 7 for a brief introduction to tools and processes for developing, assembling, and installing J2EE applications in the WebSphere for z/OS J2EE server.
- "Chapter 11. Background on migration" on page 109 for a brief introduction to migration concepts, tasks, and recommendations.

If you already have some experience with developing J2EE applications for the WebSphere for z/OS J2EE server, use Table 7 to go directly to procedures for migrating specific application components. Each procedure includes references to further details about tools, for example, in case you need additional information.

Table 7. Application migration paths at a glance

To migrate this type of application component:	From this WebSphere product:	Read the following topics:
Enterprise beans	**Advanced Edition V3.5** on distributed platforms	"Chapter 13. Migrating applications to the WebSphere for z/OS platform" on page 117
Servlets and JavaServer Pages	**Standard or Advanced Edition V3.5** on distributed platforms	"Chapter 13. Migrating applications to the WebSphere for z/OS platform" on page 117
	Standard Edition on z/OS or OS/390	"Migration scenarios for applications running on WebSphere Application Server for z/OS or OS/390 Standard Edition" on page 112
Java business objects (CORBA)	**Enterprise Edition V3.02** on z/OS or OS/390	"Migration scenarios for applications running on WebSphere Application Server for OS/390 Enterprise Edition V3.02" on page 112

Chapter 11. Background on migration

Generally speaking, migration encompasses actions that you complete to ensure that existing applications continue to function correctly on a new version or release of an IBM product. To take advantage of the J2EE server environment in WebSphere for z/OS or OS/390, however, you also might migrate existing J2EE applications from one WebSphere family environment to WebSphere for z/OS or OS/390.

Table 8 is a checklist of migration actions that might be either required or optional, depending on the type of application you want to migrate and its current run-time environment. The checklist also indicates the role of the most appropriate person to complete each action or set of actions, based on the Sun Microsystems J2EE specification. Many actions are the same as steps for developing, assembling, deploying, and installing a new application, as described in "Part 2. Creating, assembling and deploying J2EE server applications" on page 17. Familiarity with those procedures will help you complete the migration process more efficiently.

To determine which migration actions are necessary for a specific situation, refer to details in the following chapters:

- "Chapter 12. Migrating applications to a new release of WebSphere for z/OS" on page 111
- "Chapter 13. Migrating applications to the WebSphere for z/OS platform" on page 117
- "Chapter 14. Upgrading applications that are already installed in a WebSphere for z/OS J2EE server" on page 119

Table 8. Checklist of roles and potential migration actions

✓	**Potential migration actions:**	**Condition / Comments**
For system programmers or administrators:		
☐	Understand how product, system, database, or configuration changes might affect your applications	**Required** for any migration path
☐	Preconfigure J2EE resources, such as DB2	**Optional**, depending on requirements of application to be installed (required for connection pooling)
For application component providers:		

Table 8. Checklist of roles and potential migration actions (continued)

✓	**Potential migration actions:**	**Condition / Comments**
☐	Understand how tooling changes might affect your applications	**Required** because WebSphere for z/OS requires specific tools
☐	Understand how interface changes might affect your applications	**Required** because WebSphere for z/OS requires compliance with particular levels of programming specifications
☐	Change component design or rewrite source code because of interface changes	**Required**, depending on the type of application component and its compliance with required levels of programming specifications
☐	Regenerate code to pick up interface enhancements, or enhancements to or maintenance for development tools	**Optional**, as above
For application assemblers or deployers:		
☐	Repackage applications and regenerate metadata because of enhancements to or maintenance for application assembly and deployment tools	**Required** for all applications to be installed in a WebSphere for z/OS J2EE server
For application installers:		
☐	Re-install any reassembled applications to replace existing or add new EAR files and metadata	**Required** for all applications to run in a WebSphere for z/OS J2EE server
☐	Manually replace individual component files installed on z/OS or OS/390 (when application maintenance does not require repackaging EAR files or regenerating application metadata)	**Optional**
☐	Modify conversation elements because of enhancements to or maintenance for the WebSphere for z/OS Administration application	**Required** to define and activate a WebSphere for z/OS J2EE server and the J2EE resources associated with it, and to install J2EE applications
☐	Modify z/OS or OS/390 constructs because of product, system, or configuration changes. Such changes include moving or reconstructing application databases, changing WLM service goals, and so on.	**Required** for: • Applications that require role-based security • Web applications that formerly ran in the WebSphere for z/OS Standard Edition environment

Chapter 12. Migrating applications to a new release of WebSphere for z/OS

A new release of the WebSphere for z/OS or OS/390 product, or its associated tools, might require you to migrate applications that are already installed and running on a previous release. You might have to migrate these applications to either adapt to or exploit enhancements or service updates in the new version or release. (If you want to know how to migrate applications that currently run in another WebSphere family environment, see “Chapter 13. Migrating applications to the WebSphere for z/OS platform” on page 117.)

The following table shows the subtasks and associated information for learning about a new release of WebSphere for z/OS, and for migrating applications that currently run on a previous release:

Subtask	**Associated information (See . . .)**
Determine which migration paths are supported	The migration roadmap in *WebSphere Application Server V4.0 for z/OS and OS/390: Installation and Customization*, GA22-7834
Determine what changes have been introduced with a particular version or release	The release summary for the new WebSphere for z/OS product. Release summaries are listed in *WebSphere Application Server V4.0 for z/OS and OS/390: Installation and Customization*.
Understand how each change might affect your installation and its existing applications	The change descriptions listed in a particular release summary (in *WebSphere Application Server V4.0 for z/OS and OS/390: Installation and Customization*). These descriptions of functional updates summarize the change, the areas of processing that might be affected, migration tasks, new and changed interfaces, and sources of additional details.
Determine how to migrate existing applications to a new release	• “Migration scenarios for applications running on WebSphere Application Server for z/OS or OS/390 Standard Edition” on page 112 • “Migration scenarios for applications running on WebSphere Application Server for OS/390 Enterprise Edition V3.02” on page 112

Migration scenarios for applications running on WebSphere Application Server for z/OS or OS/390 Standard Edition

One of the WebSphere products your installation might be using for Web serving is the WebSphere Application Server for z/OS or OS/390 Standard Edition. Depending on the version of the Standard Edition product you are currently using, you might have to upgrade elements of your z/OS or OS/390 system as well as perform migration actions for your Web applications.

For additional information about the Standard Edition products and WebSphere for z/OS, see:

- "Appendix C. Migration considerations for Web applications running on WebSphere Application Server Standard Edition" on page 161 for information about migrating from Standard Edition V3.02 or V3.5
- "Migrating from version 3.5" on page 161, if you want to run existing Web applications in a Standard Edition environment at the same time you run new Web applications in a WebSphere for z/OS J2EE server.
- *WebSphere Application Server for OS/390 Application Server Planning, Installing and Using, Version 3.5*, GC34–4835, to determine how to:
  - Install and configure Standard Edition V3.5.
  - Migrate Web applications from previous versions of Standard Edition to Standard Edition V3.5.

Migration scenarios for applications running on WebSphere Application Server for OS/390 Enterprise Edition V3.02

Table 9 on page 113 lists the possible migration scenarios for applications that currently run on WebSphere Application Server for OS/390 Enterprise Edition V3.02. If you need further details, use the following information sources:

- Sun Microsystems web site (`java.sun.com`) for details about differences between levels of the Software Development Kit (SDK), and between Enterprise bean specifications.
- "Part 2. Creating, assembling and deploying J2EE server applications" on page 17 if you need instructions or guidelines for tasks related to creating, assembling, and installing applications in a WebSphere for z/OS J2EE server.
- *WebSphere Application Server V4.0 for z/OS and OS/390: Assembling CORBA Applications*, SA22-7848 if you need instructions or guidelines for tasks related to creating, assembling, and installing CORBA applications in a WebSphere for z/OS MOFW server.

Note: You **cannot** install the WebSphere for z/OS J2EE server on the same z/OS or OS/390 system image, or in the same sysplex, on which you are currently running WebSphere Application Server Enterprise Edition for OS/390 V3.02.

Table 9. Summary of migration tasks for WebSphere Application Server for OS/390 Enterprise Edition V3.02 applications

Migration scenario:	**Comments:**
Migrate an Enterprise Edition V3.02 Enterprise bean to V4.0	**Recommendation:** Because the V3.02 support for Enterprise beans was limited, consider starting over— design and code new Enterprise beans to match the new support that the V4.0 J2EE server provides. **Rule:** V4.0 of WebSphere for z/OS supports two types of servers: the new J2EE server, and servers that are based on the V3.02 managed-object framework (MOFW) for CORBA applications. You cannot run Enterprise beans in a V4.0 (MOFW) server. If you must preserve a bean from V3.02 and upgrade it for the V4.0 J2EE server support, do the following: 1. Import the bean source into an appropriate development tool. **Note:** If you do not have access to source code, you may import a bean JAR file to check for deprecated interfaces. If there are none, you can skip to Step 4. 2. **Recommendation:** Rewrite code to eliminate functions, if any, that are deprecated because of the new SDK level or EJB specification. 3. For entity beans, review requirements for container-managed or bean-managed persistence 4. Regenerate bean code and export the resulting JAR file. 5. Use the WebSphere for z/OS Application Assembly tool to import the JAR file and package the bean into an Enterprise archive (EAR) file. 6. Use the WebSphere for z/OS Administration application to install the EAR file.

Table 9. Summary of migration tasks for WebSphere Application Server for OS/390 Enterprise Edition V3.02 applications (continued)

Migration scenario:	**Comments:**
Migrate an Enterprise Edition V3.02 Java business object to a V4.0 Enterprise bean	**Recommendation:** Follow this migration path only if you purposely designed and coded them according to guidelines supplied in *WebSphere Application Server Enterprise Edition for OS/390 Component Broker Assembling Applications Guide Version 3.02*, GA22-7326, with the intent of later converting the Java BOs to Enterprise beans. Otherwise, the conversion effort will probably be more complicated and difficult than coding a new Enterprise bean from scratch. To convert a Java BO to an Enterprise bean: 1. Understand the differences between the CORBA and EJB programming models and the new SDK level. 2. Create a new Enterprise bean in an appropriate development tool. Copy the Java BO's business logic and modify it as necessary: • **Recommendation:** Rewrite code to eliminate functions, if any, that are deprecated because of the new SDK level or EJB specification. • For entity beans, review requirements for container-managed or bean-managed persistence • Rewrite other code as necessary, paying particular attention to the following: – Implementation inheritance – Helper classes that deal with other objects or with CORBA services – Code containing references to properties that may vary based on environment – Association code for objects that have associations with other objects 3. Generate bean code and export the resulting JAR file. 4. Use the WebSphere for z/OS Application Assembly tool to import the JAR file and package the bean into an Enterprise archive (EAR) file. 5. Use the WebSphere for z/OS Administration application to install the EAR file.
Migrate an Enterprise Edition V3.02 Java business object to run in a V4.0 (MOFW) server	To upgrade a Java BO to run in a V4.0 server, do the following: 1. Import the source into an appropriate development tool. **Rule:** Use Object Builder 3.5 for your Java BOs. 2. **Recommendation:** Rewrite code to eliminate functions, if any, that are deprecated because of the new SDK level. 3. If this object contains code that used to drive Enterprise beans in a Enterprise Edition V3.02 MOFW server, you need to change the code that provides the following functions. Enterprise beans must reside in V4.0 J2EE server now, so the programming model for using beans has changed. • Setting initial context • Finding and using homes • Narrowing object references 4. Regenerate code and export the resulting JAR file. 5. Transfer the application files to z/OS or OS/390, and recompile the code by running the `all.mak` file. 6. Use the WebSphere for z/OS Administration application to install the application in a V4.0 (MOFW) server.

Table 9. Summary of migration tasks for WebSphere Application Server for OS/390 Enterprise Edition V3.02 applications (continued)

Migration scenario:	**Comments:**
Migrate an Enterprise Edition V3.02 C++ business object to run in a V4.0 (MOFW) server	You do not have to change the source code of existing V3.02 C++ business objects at all, but you might, however, consider taking them back through the application development tooling to take advantage of any service updates or enhancements to the compiler or Object Builder. **Rules:** • Use Object Builder 3.5 for your C++ business objects. • You must recompile your C++ business objects on z/OS or OS/390 To upgrade C++ business objects to run in a V4.0 server, do the following: • Rerun the `all.mak` file to recompile the code on z/OS or OS/390. • Use the Administration application to install the application in a V4.0 (MOFW) server.
Migrate a ″native″ z/OS or OS/390 client application that drives CORBA server applications	You do not have to change existing C++ or Java clients that use the CORBA programming model to use CORBA server applications. You might, however, consider taking them back through the application development tooling to take advantage of any service updates or enhancements to the compiler or tooling.
Migrate a ″native″ z/OS or OS/390 client application that drives Enterprise beans	Because Enterprise beans must reside in J2EE servers, the programming model for using beans has changed. If the ″native″ z/OS or OS/390 client application is using the CORBA programming model to access beans, you must change the source code. Use the following procedure to do so: 1. Import the source into an appropriate development tool. 2. **Recommendation:** Rewrite code to eliminate functions, if any, that are deprecated because of the new SDK level. 3. Change the code that provides the following functions: • Setting initial context • Finding and using homes • Narrowing object references 4. Regenerate code and export the resulting JAR file. 5. Recompile the code on z/OS or OS/390.
Migrate a distributed client application	You do not have to change clients that meet both of the following criteria: • They use C++ or Java business objects installed a MOFW server, and • They run in WebSphere Application Server Enterprise Edition Component Broker Versions 3.0, V3.5, or V3.6, on a workstation platform (Windows NT, and so on).

Chapter 13. Migrating applications to the WebSphere for z/OS platform

With previous releases of WebSphere family products, programming model differences inhibited the portability of applications from distributed platforms to OS/390. With the support available through the WebSphere for z/OS J2EE server, however, moving J2EE application components from other WebSphere environments to WebSphere for z/OS or OS/390 should be relatively easy. In fact, the primary considerations for migrating J2EE application components are:

1. Differences between supported specification levels.

   For example, WebSphere for z/OS J2EE server supports the Sun Microsystems Java Servlet 2.2 specification. If you have a V2.2 servlet running in another WebSphere environment, you can migrate it without code changes. If you have a V2.1 servlet, however, you need to understand the differences between the V2.1 and V2.2 specifications, and remove any deprecated functions from your servlet code.

   For information about migrating components to the appropriate specification levels, use the instructions available through the **InfoCenter** at `http://www.ibm.com/software/webservers/appserv/library.html`
2. Differences between database requirements or access.

   For example, if you want to migrate Enterprise beans that use container-managed persistence, you do not have to make any changes if you are using DB2 as a persistent datastore. If your applications directly access DB2, however, you will need to account for differences between DB2 on distributed platforms and DB2 on z/OS or OS/390. These differences mean that you need to change JDBC driver names and some SQL syntax.

   For further details, see *z/OS and OS/390 DB2 Application Development Guide*, which is located at `http://www.ibm.com/software/db2os390/v7books.html`
3. Platform-specific tools for assembly, deployment, and installation.

   WebSphere for z/OS currently requires the use of its Application Assembly tool to prepare J2EE applications for installation, and the use of its Administration application to install these applications. You will have to use these two tools as part of the migration process.

Table 10 on page 118 lists the possible migration scenarios for applications that currently run in other WebSphere family environments. If you need further details, use the following information sources:

- See the Sun Microsystems web site (`java.sun.com`) for details about differences between levels of the Software Development Kit (SDK), and between specification levels for J2EE application components.
- See "Part 2. Creating, assembling and deploying J2EE server applications" on page 17 if you need instructions or guidelines for tasks related to creating, assembling, and installing applications in a WebSphere for z/OS J2EE server.

Table 10. Summary of migration scenarios for applications in other WebSphere family environments

Migration scenario:	**Comments:**
Moving servlets and JSPs from WebSphere Application Server Standard Edition V3.5	1. Upgrade application development software on your workstation, if necessary. 2. Import your application component into an appropriate development tool. Then: • Rewrite code to eliminate functions that are deprecated because of new SDK levels or component specification levels. • If your Web components use J2EE resources, such as DB2, review the code and make any appropriate changes. 3. Regenerate component code and create a Web archive (WAR) file for the Web application. 4. Export the resulting WAR file from the application development tool. 5. Use the WebSphere for z/OS Application Assembly tool to import the component files and package the component into an Enterprise archive (EAR) file. 6. Use the WebSphere for z/OS Administration application to install the EAR file.
Moving Enterprise beans, servlets and JSPs from WebSphere Application Server Advanced Edition V3.5	1. Upgrade application development software on your workstation, if necessary. 2. Import your application component into an appropriate development tool. Then: • Rewrite code to eliminate functions that are deprecated because of new SDK levels or component specification levels. • For entity beans, review requirements for container-managed or bean-managed persistence. • If your Web components use J2EE resources, such as DB2, review the code and make any appropriate changes. 3. Regenerate component code and create a Web archive (WAR) file for the Web application components, or JAR files for Enterprise beans. 4. Export the resulting WAR or JAR files from the application development tool. 5. Use the WebSphere for z/OS Application Assembly tool to import the component files and package the components into an Enterprise archive (EAR) file. 6. Use the WebSphere for z/OS Administration application to install the EAR file. 7. Review JNDI lookup names for the Enterprise bean homes used in your components, and make sure the lookup names match the names under which the required components are registered on z/OS.

Chapter 14. Upgrading applications that are already installed in a WebSphere for z/OS J2EE server

The following notes apply for working with applications that you have previously installed in a WebSphere for z/OS J2EE server:

Notes:

1. You may install multiple versions of the same application in the same J2EE server, as long as each application version is unquely named, separately packaged and installed through the WebSphere for z/OS Application Assembly tool. If each version of the application requires a different database schema, you may define a separate J2EE resource connection for each application.
2. If you make changes to an already installed application, and those changes are minor enough that they do not require you to reassemble and reinstall the application using the appropriate tools, you may:
   a. Replace JAR or WAR files in the z/OS or OS/390 HFS
   b. Issue a z/OS or OS/390 workload manager command to drain current work
   c. Start new server regions

   Use this process as a replacement for the Standard Edition servlet reloading function.

Part 5. Appendixes

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 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. The default is /WebSphere390/CB390.

Recommendation: The System Management server region user ID (CBSYMSR1 in our BBOCBRAC sample) should be the owner of the /WebSphere390/CB390 directory. The System Management server region writes files to this directory. The permission bits should be 775 so other server region user IDs have read access to the directory.

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, you must modify an initial environment file, which the bootstrap job uses. This is the only time you should modify an environment file directly.

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. Defining environment data by coding environment variables prior to the bootstrap process. In this situation, you modify the sample we give you. The bootstrap job reads the file and places the environmental data into the system management database. This is the only time you modify an environment file directly in the HFS.

   For the syntax of the environment variables, see "Environment variable syntax" on page 126.
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 BBOASRIA 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 OS/390 clients

The Administration application does not manage environment variables for OS/390 clients. You must create and manage OS/390 client environment files and point to them from client programs. Table 11 on page 128 tells you which environment variables are required or optional for 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.

Environment variable use

Not all environment variables need to be used for each server or client. Table 11 on page 128 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 11. Where to use environment variables

Environment variable=<default>	**Daemon server instance**	**System Management server instance**		**Naming server instance**		**Interface Repository instance**		**J2EE server instance**		**OS/390 client**
	control region	**control region**	**server region**	**control region**	**server region**	**control region**	**server region**	**control region**	**server region**	
BBOLANG=ENUS	O	O	O	O	O	O	O	O	O	O
CBCONFIG= /WebSphere390/CB390	R	R	R	R	R	R	R	R	R	
CLASSPATH=			O		O		O		O[1]	
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=										
com.ibm.ws.naming.ldap.containerdn= <ibm-wsnTree=t1,o=<org>, c=<country>>				O	O					
com.ibm.ws.naming.ldap. domainname=*domain name*				O	O					
com.ibm.ws.naming.ldap.masterurl= ldap://<ip name>:<port>				O	O					
DAEMON_IPNAME=	R	O								
DAEMON_PORT=5555	O[2]	O[2]								
DEFAULT_CLIENT_XML_PATH=										O[3]
DM_GENERIC_SERVER_NAME= CBDAEMON	O[2]	O[2]								
DM_SPECIFIC_SERVER_NAME= DAEMON01	O[4]	O[4]		O[4]		O[4]		O[4]		

Table 11. 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		OS/390 client
	control region	control region	server region	control region	server region	control region	server region	control region	server region	
HOME=										O
IBM_OMGSSL=0								O	O	
ICU_DATA= /usr/lpp/WebSphere/bin/		R	R							
IR_GENERIC_SERVER_NAME= CBINTFRP		O								
IR_SPECIFIC_SERVER_NAME= INTFRP01	O[4]	O[4]		O[4]		O[4]		O[4]		
IRPROC=BBOIR	O	O								
IVB_DRIVER_PATH= /usr/lpp/WebSphere		R	R							
java.naming.security.credentials= <password>		O	O							
java.naming.security.principal= <userid>		O	O							
JAVA_COMPILER=									O	O
JAVA_IEEE754=										O[11]
JVM_BOOTCLASSPATH=			O						O	
JVM_BOOTLIBRARYPATH=			O						O	
JVM_DEBUG=			O						O	
JVM_ENABLE_CLASS_GC=			O						O	
JVM_ENABLE_VERBOSE_GC=			O						O	
JVM_EXTRA_OPTIONS=									O	

Table 11. 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		OS/390 client
	control region	control region	server region	control region	server region	control region	server region	control region	server region	
JVM_HEAPSIZE=256									O	
JVM_LOCALREFS=			O						O	
JVM_LOGFILE=									O	O
JVM_MINHEAPSIZE=			O						O	
LDAPBINDPW=		F			R[5]					
LDAPCONF=		F			R[5]					
LDAPHOSTNAME=		F			R[5]					
LDAPIRBINDPW=		F					R[6]			
LDAPIRCONF=		F					R[6]			
LDAPIRHOSTNAME=		F					R[6]			
LDAPIRNAME=		F					R[6]			
LDAPIRROOT=		F					R			
LDAPNAME=		F			R[5]					
LDAPROOT=		F			R					
LIBPATH=			O		O		O		O[1]	
LOGSTREAMNAME=	O	O								
MIN_SRS=[0 for MOFW, 1 for J2EE]								O		
NM_GENERIC_SERVER_NAME= CBNAMING		O								
NM_SPECIFIC_SERVER_NAME= NAMING01	O[4]	O[4]		O[4]		O[4]		O[4]		

Table 11. 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**		**OS/390 client**
	control region	**control region**	**server region**	**control region**	**server region**	**control region**	**server region**	**control region**	**server region**	
NMPROC=BBONM	O	O								
OTS_DEFAULT_TIMEOUT=30	O	O	O	O	O	O	O	O	O	
OTS_MAXIMUM_TIMEOUT=60	O	O	O	O	O	O	O	O	O	
PATH=									O	O
RAS_MINORCODEDEFAULT= NODIAGNOSTICDATA										
REM_DCEPASSWORD=										O
REM_DCEPRINCIPAL=										O
REM_PASSWORD=		O[7]		O[7]		O[7]		O[7]		O
REM_USERID=		O[7]		O[7]		O[7]		O[7]		O
RESOLVE_IPNAME=		O[8]	O[9]	O[9]	O[9]	O[9]	O[9]	O[9]	O[9]	R[10]
RESOLVE_PORT=900		O	O	O	O	O	O	O	O	O
SM_DEFAULT_ADMIN= CBADMIN		O								
SM_GENERIC_SERVER_NAME= CBSYSMGT		O								
SM_SPECIFIC_SERVER_NAME= SYSMGT01	O[4]	O[4]		O[4]		O[4]		O[4]		
SMPROC=BBOSMS	O	O								
SOMOOSQL=									O	
SRVIPADDR=	O	O		O		O		O		
SSL_KEYRING=										O
SYS_DB2_SUB_SYSTEM_NAME=DB2	R	R		R		R		R		

Table 11. 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		OS/390 client
	control region	control region	server region	control region	server region	control region	server region	control region	server region	
TRACEALL=1	O	O	O	O	O	O	O	O	O	O
TRACEBASIC=	O	O	O	O	O	O	O	O	O	O
TRACEBUFFCOUNT=4	O	O	O	O	O	O	O	O	O	
TRACEBUFFLOC=(Server: BUFFER Client: SYSPRINT)	O	O	O	O	O	O	O	O	O	O
TRACEBUFFSIZE=1M	O	O	O	O	O	O	O	O	O	
TRACEDETAIL=	O	O	O	O	O	O	O	O	O	O
TRACEMINORCODE=										
TRACEPARM=00	O									

Table 11. 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**		**OS/390 client**
	control region	**control region**	**server region**	**control region**	**server region**	**control region**	**server region**	**control region**	**server region**	

Notes:

1. Required for server regions that use Java, including the IMS PAA and CICS PAA.
2. If you specify a value for the Daemon Server, you must provide the same value for the System Management Server control region.
3. Required when the client uses the System Management Scripting API.
4. You must specify this for the second and subsequent systems in a sysplex.
5. LDAPCONF is mutually exclusive with LDAPBINDPW, LDAPHOSTNAME, and LDAPNAME. Either LDAPCONF is required, or LDAPBINDPW, LDAPHOSTNAME, and LDAPNAME are required.
6. LDAPIRCONF is mutually exclusive with LDAPIRBINDPW, LDAPIRHOSTNAME, and LDAPIRNAME. Either LDAPIRCONF is required, or LDAPIRBINDPW, LDAPIRHOSTNAME, and LDAPIRNAME are required.
7. Used when a server becomes a remote client of another server.
8. Default is the value of DAEMON_IPNAME during bootstrap.
9. Default is the local system IP name. Generally, do not code.
10. Optional if a Daemon Server is on the same system as the client, in which case the default is the local system IP name.
11. Required for Java clients that run on OS/390.

Environment variable descriptions

BBOLANG=*LANGUAGE*
The name of the WebSphere for z/OS message catalog used. The default is ENUS.

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`

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 OS/390 client to apply to the current transaction flows. Normally, you would set DCE security for an 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 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 OS/390 client ORB writes error information.

Example: `CLIENTLOGSTREAMNAME=MY.CLIENT.ERROR.LOG`

CLIENT_RESOLVE_IPNAME=*IP_NAME*
The Internet Protocol name that an 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. The values are in integers and signify the time in tenths of seconds (thus, a value of 10 is 1 second). The default value is 0, which means no time-out value is set.

Example: `CLIENT_TIMEOUT=20`

| **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. For 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
```

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`

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 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`

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`

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`

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. For C++ or Java clients running on OS/390, set this variable to `/tmp` when debugging business objects with the IBM Distributed Debugger.

Example: `HOME=/tmp`

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/`

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_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`

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 OS/390 uses the JIT compiler, so you do not have to explicitly set this environment variable. If you are debugging Java business objects, 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 OS/390 run. This environment variable setting is required only for Java clients that run on OS/390.

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.

| **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 JVM messages to SYSOUT for debugging purposes. Set JVM_DEBUG=1 to invoke JVM messaging.

| **JVM_ENABLE_CLASS_GC=1**
Enables class objects to be garbage collected. The value `1` is required for enabling class object garbage collection. This option is equivalent to the `-Xnoclassgc` Java invocation option.

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.

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 messages from the JVM will be logged.

Recommendation: 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 JVM messages to the SYSOUT for a specific server.

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_install_path/lib:/usr/lpp/java/J1.3/bin:/usr/lpp/java/J1.3/bin/classic:/usr/lpp/WebSphere/lib
```

where *db2_install_path* is the HFS where you installed DB2 for OS/390.

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.

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. When tracing and debugging Java on OS/390, for the application server only, include the path to the executable called irmtdbgj.

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 UNIX System Services 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.

REM_DCEPASSWORD=*password*
The password of the remote DCE principal passed in the security context when an 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 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 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`

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 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_KEYRING=*keyring*
The name of the OS/390 client's key ring used in SSL processing. This key ring must reside in RACF.

Example: `SSL_KEYRING=IVPRING`

SYS_DB2_SUB_SYSTEM_NAME=*NAME*
The DB2 for OS/390 name used by Daemon and System Management servers to connect to the database. Use either the DB2 for OS/390 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 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`

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 do not 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.

JVM property use

Table 12 on page 152 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 12. Where to use JVM properties

JVM property=<default>	**J2EE server instance**	**MOFW server instance**
	server region	**server region**
com.ibm.ws390.trace.settings=	O	O
com.ibm.ws390.wc.config.webcontainer.configfile=	O	

Properties descriptions

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 99, and
- *WebSphere Application Server V4.0 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.webcontainer.configfile=*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.

For information about using your own Web container configuration file, see the documentation about enabling Web application support in WebSphere for z/OS, available at
`http://www.ibm.com/software/webservers/appserv/`

Example:

```
com.ibm.ws390.webcontainer.configfile=/usr/lpp/WebSphere/AppServer/bin/WebCon2.conf
```

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=4.00
# ================================================================== #
#
#  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. For more information, please
#  read WebSphere Application Server V4.0 for z/OS and OS/390:
#  Assembling 2EE Applications
#
#  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
#       =================
#       - Http Session Tracking
#       - Virtual Host
#
#       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.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
#        Web container attempts to use the session services,
#        an exception will be thrown.
#
#        The default is true.
#
#
session.enable=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.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 "sesessionid".
#
#
session.cookie.name=sesessionid
#
#---------------------------------------------------------------------#
#
#      session.cookie.comment=<comment>
#
#        The value of this property is a string that specifies
#        a comment about the cookie, if cookies are enabled.
#
#        The default is "WebSphere Session Support".
#
#
session.cookie.comment=servlet 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.path=<path>
#
#        The value of this property is a string that specifies
#        the path field that will be sent for session cookies.
#        Specify a value only to restrict to which paths on the
#        server (and, therefore, to which servlets, JHTML files,
#        and HTML files) the cookies will be sent.
#
#        Specifying "/" for the path indicates the root directory,
#        which means that the cookie will be sent on any access to
#        the given server.
#
#        The initial setting is "/".
#
#
session.cookie.path=/
#
#---------------------------------------------------------------------#
#
#      session.cookie.secure=true|false
#
#        The value of this property is a boolean that
#        indicates whether session cookies include the secure
#        field. If this property is set to "true", this will
#        restrict the exchange of cookies to only HTTPS
#        sessions. Otherwise, they will be exchanged in
#        HTTP sessions as well.
#
#        The default is false.
#
#
session.cookie.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.
#
#
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.datasourcename=<name>
#
#        Specifies name of the datasource that is to be used to obtain
#        a connection to the database where the session data will be stored.
#        The name specified should be the same name that your
#        application will use to perform the naming service lookup
#        on the datasource object.
#
#       The default name is jdbc/SessionDataSource.
#
#
session.datasourcename=jdbc/SessionDataSource
#
#---------------------------------------------------------------------#
#
#      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.domain
#
#        Specifies the domain name for which the session cookie is
#        valid.
#
#        The default is null.
#
#
session.domain=
#
.# ================================================================== #
#
#   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.
#        There can be multiple alias properties per virtual host.
#
#        The Application Server supports a special hostname, "localhost",
#        which maps all requests to the virtual host definition.
#        This support is provided for the initial verification program.
#        IBM recommends that it not be used beyond that purpose.
#
#        There is no default.
#
#
```

```
host.default_host.alias=localhost
#
#---------------------------------------------------------------------#
#
#      host.<virtual_hostname>.contextroots=<path_prefix>
#
#      Specifies the path prefix associated with the ServletContext that
#      servlets in this virtual host are a part of. A servlet uses this
#      path to access available resources. Using this path, a servlet can
#      log events, obtain URL references to resources, and set and store
#      attributes that other servlets in the context can use.
#
#      A ServletContext is rooted at a specific path within a Web container.
#      If this context is the ôdefaultö context rooted at the base of the
#      Web containerÆs URL namespace, this path will be an empty string.
#      Otherwise, this path starts with aÆ/Æ character but does not
#      end with aÆ/Æ character.
#
#      The default is /*.
#
#
<ahost.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=
#
# ================================================================== #
#
#   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.path
#            session.cookie.secure
#            session.tablesize
#            session.invalidationtime
#            session.tableoverflowenable
#            session.dbenable
#            session.dbtablename
#            session.domain
#
#     Additionaly, you must provide a value for the new virtual host property,
#     session.datasourcename, or use the default value jdbc/SessionDataSource.
#
#      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. Migration considerations for Web applications running on WebSphere Application Server Standard Edition

Migrating from version 3.5

WebSphere Application Server Standard Edition V3.5 can co-exist on the same z/OS or OS/390 system with WebSphere Application Server for z/OS or OS/390 V4.0 as long as the V3.5 HFS is mounted on a different mount point than the V4.0 HFS. The ability to have both the V3.5 and V4.0 Application Server on the same system enables you to migrate existing V3.5 Web applications to your Web container over time, while creating new applications in a WAR file format that can be installed into the V4.0 Web container. Both sets of applications can be accessed, using HTTP protocol, from a browser. This capability enables you to:

- Continue to run existing V3.5 Web applications while becoming familiar with V4.0.
- Develop new Web applications at the Java Servlet Specification Version 2.2 level, package them as WAR files, and install them in a Web container on the J2EE server.
- Slowly migrate existing V3.5 Web applications to a Web container.
- Continue to run Web applications that do not comply with the Java Servlet Specification Version 2.2, that or require JavaServer Pages (JSPs) written at a 0.91 or 1.0 specification level, on a V3.5 Application Server.

To continue using your V3.5 Application Server, you must:

- Specify the fully qualified name of the V3.5 was.conf file as the second parameter on the ServerInit directive in the HTTP server's httpd.conf configuration file that indicates the entry point to the V4.0 WebSphere for z/OS plugin's initialization routine.
- Change the value specified for the appserver.version property in the V3.5 was.conf file from 3.50 to 4.00.

If the HTTP Server detects a value in this second position of the ServerInit directive when it receives a request from a browser, it:

1. Searches the V3.5 was.conf file for a **deployedwebapp** property for the requested application. If a match is found, processing will be handled by the V3.5 Application Server.

2. If a matching **deployedwebapp** property is not found, the HTTP server sends the request to the J2EE server for processing. The J2EE server then searches the appropriate Web and EJB containers for the requested application.

If a second parameter is not specified on the ServerInit directive, all requests will be sent directly to a J2EE server for processing.

When you are ready to migrate your Web applications to a Web container, you must:

1. Ensure that all of the servlets and JSPs contained in your Web applications conform to the Javasoft Servlet Specification V2.2 and the JavaServer Pages 1.1 specification level.
2. For each application, package all of the Web components into a WAR file, using standard Java Archive tools (see "Migrating Web applications to WAR files" on page 166).
3. Using the Application Assembly Tool for z/OS and OS/390 that is shipped with the V4.0 product, convert each WAR file to an EAR file and install it into a Web Container on the V4.0 J2EE server.

Migrating from V3.02

You have two options for migrating from V3.02:

1. You can first migrate to V3.5 and then to V4.0.

   Migrating to V3.5 and then to V4.0 enables you to continue using Web applications written to the V2.1 Javasoft Servlet Specification and JavaServer Pages written to the 0.91 and 1.0 specification levels, provided you configure your V3.5 Application Server to run in compatibility mode. (See the *WebSphere Application Server for OS/390 Application Server Planning, Installing and Using, Version 3.5*, GC34–4835 for more information about how to migrate from V3.02 and how to set up your V3.5 Application Server to run in compatibility mode.)

   Once you have your V3.5 Application Server running, you can follow the instructions in the previous section, "Migrating from version 3.5" on page 161, and add V4.0 to the same z/OS or OS/390 system. You can now run your current Web applications on the V3.5 Application Server while developing new applications for the V4.0 Application Server.
2. You can migrate directly to V4.0. To migrate directly to V4.0, you must:
   a. Ensure that all of the servlets and JSPs contained in your Web applications conform to the Javasoft Servlet Specification V2.2 and the JavaServer Pages 1.1 specification level.
   b. For each application, package all of the Web components into a WAR files, using standard Java Archive tools (see "Migrating Web applications to WAR files" on page 166).

c. Using the Application Assembly Tool for z/OS and OS/390 that is shipped with the V4.0 product, repackage each WAR file as an EAR file and install it into a Web container on the V4.0 J2EE server.

Migrating from JDK 1.1x to SDK 1.3

Regardless of whether you migrate directly to V4.0 or migrate to V3.5 first, both the V4.0 and V3.5 Application Server run-time environments are built on SDK 1.3. You should be able to run most programs that ran under JDK 1.1x with little or no modification. However, the following list summarizes some minor potential incompatibilities that may require your applications to be modified:

1. There are now two Timer classes:
   - java.util.Timer (new)
   - javax.swing.Timer (existed in V1.1x)

   If an application has the following two import statements:
   ```
   import java.util.*;
   import javax.swing.*;
   ```
   and refers to javax.swing.Timer by its unqualified name, the following import statement must be added in order for the ambiguous reference to class Timer to be correctly resolved:
   ```
   import javax.swing.Timer;
   ```
2. The implementation of method **java.lang.Double.hashcode** has been changed to conform to the API specification. This change should not affect the behavior of existing applications because hashcode returns a truncated integer value.
3. A new Permission class, **java.sql.SQLPermission**, has been added in version 1.3. WebSphere Application Server V3.5 on MultiPlatforms supports this new class; WebSphere Application Server for OS/390 V3.5 does not.
4. The internal implementation of the Java Sound APIs (in class **com.sun.media.sound.SimpleInputDevice**) now checks **javax.sound.sampled.AudioPermission**. This new check means that, under 1.3, applets must now be given the appropriate AudioPermission to access audio system resources
5. JInternalFrames are no longer visible by default. Developers must set the visibility of each JInternalFrame to true in order to have it show up on the screen.
6. The **TableColumn.getHeaderRenderer** method returns null by default. Therefore, you must use the new **JTableHeader.getDefaultRenderer** method instead to get the default header renderer.

7. The JTable's default text editor now gives setValueAt objects of the appropriate type, instead of always specifying strings. For example, if setValueAt is invoked for an Integer cell, then the value is specified as an Integer instead of a String. If you implemented a table model, you might have to change its **setValueAt** method to take the new data type into account. If you implemented a class that is used as a data type for cells, make sure that your class has a constructor that takes a single String argument.
8. It is no longer possible for sufficiently trusted code to modify final fields by first calling **Field.setAccessible(true)** and then calling **Field.set()**. An IllegalArgumentException will be thrown when an attempt is made to modify a final field. The JNI Set<Field> routines can be used to set non-static final fields.
9. The specification and behavior of the constructors **BasicPermission(String name)** and **BasicPermission(String name, String actions)** in class **java.security.BasicPermission** have been modified. When the name parameter is null, the constructors now throw a NullPointerException. When name is an empty string, the constructors now throw an IllegalArgumentException. This change of behavior is inherited by subclasses of **BasicPermission**. The change also affects the behavior of **java.lang.System.getProperty()** and **java.lang.System.setProperty()** whose implementations construct an instance of **PropertyPermission**, a subclass of **BasicPermission**. Because of this change, a call to **getProperty** or **setProperty** with an empty property name (that is, **getProperty("")** or **setProperty("", value)**) will result in an IllegalArgumentException. When using JDK instead of SDK, such a call would return quietly with no exception.
10. The behavior of **java.net.URL** has changed for cases where a URL instance is constructed from a String. A final slash ('/') is not automatically added to a URL when the URL is constructed without one. For example, the following code:

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

    now results in the following output:

    ```
    http://www.xxx.yyy
    ```
11. The javac complier has a new implementation with the following implications:
    - Inherited members of an enclosing class are now accessible.
    - A local variable or catch clause parameter can be hidden when it is declared within the scope of a like-named method parameter, local variable, or catch clause parameter.
    - It is now illegal for a package to contain a class or interface type and a subpackage with the same name. A package, class, or interface is

presumed to exist if there is a corresponding directory, source file, or class file accessible on the classpath or the sourcepath, regardless of its content.
  - A qualified name in a constant expression must be of the form TypeName.identifier.
  - Member classes of interfaces are inherited by implementing classes

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

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

`http://www.ibm.com/s390/java/`

Setting runtime properties

In V3.5 of the Application Server, runtime settings, such as the location of the JVM properties file, the level of logging that is to be performed, and the location of the working directory, were set in the was.conf file. In V4.0, the runtime settings established for the J2EE server configuration apply to the containers within that server. Therefore, properties, such as the **appserver.jvmpropertiesfile** and **appserver.loglevel** properties, do not exist in the webcontainer.conf file.

Setting Session properties

You can continue to use most of the session settings you had in effect in V3.x of the Application Server. The following session properties can be copied from your V3.x was.conf file and added to the V4.0 Web container configuration file, webcontainer.conf:

- session.enable
- session.urlrewriting.enable
- session.cookies.enable
- session.protocolswitchrewriting.enable
- session.cookie.name
- session.cookie.comment

- session.cookie.maxage
- session.cookie.path
- session.cookie.secure
- session.tablesize
- session.invalidationtime
- session.tableoverflowenable
- session.dbenable
- session.dbtablename
- session.domain

Accessing services

In V3.5 of the Application Server, access to services such as JDBC and JNDI, was established through settings in the was.conf file. In V4.0, access to these tools is provided by the J2EE server. Therefore, properties, such as the **jdbcconnpool** properties, do not exist in the webcontainer.conf file.

Migrating Web applications to WAR files

When you are ready to convert your V3.5 Web applications to WAR files, use a conversion tool, such as the IBM WebSphere Studio product, to convert your Web applications into WAR files.

Note: If your Web application contains more than one servlet, after you have converted your application to a WAR file, check the web.xml file that the tool will also create, to make sure that all of the <servlet> XML tags are grouped together; not intermixed with the <servlet-mapping> tags. Some tools intermix the <servlet>tags with <servlet-mapping> tags, which can create processing errors. If the tags are intermixed, edit this file and group all of the <servlet> tags together and group all of the <servlet-mapping> tags together.

Servlet reloading

The servlet reloading function that existed in previous versions of the Application Server is no longer supported. WLM commands are now used to refresh servlets without causing an interruption of service.

Serving servlets by class name

Servlets can no longer be served by using their class name. Class names must be mapped to a servlet in a WAR file.

Appendix D. 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

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
CICS
DB2
IBM
IMS
IMS/ESA
Language Environment
Open Class
OS/390
RACF
VisualAge
VTAM
WebSphere
z/OS

Glossary

For more information on terms used in this book, refer to one of the following sources:

- *WebSphere Application Server V4.0 for z/OS and OS/390 Glossary*, SC09-4450, located on the Internet at:

  `http://www.ibm.com/software/webservers/appserv/`
- Sun Microsystems Glossary of Java Technology-Related Terms, located on the Internet at:

  `http://java.sun.com/docs/glossary.html`

If you do not find the term you are looking for, refer to *IBM Glossary of Computing Terms*, located on the Internet at:

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

or the Sun Web site, located on the Internet at:

`http://www.sun.com/`

Index

Program Number: 5655-F31

Printed in the United States of America

SA22-7836-01