WebSphere Application Server for z/OS and OS/390

Enabling Web Applications on a J2EE Server

Version 4.0

WebSphere Application Server for z/OS and OS/390

Enabling Web Applications on a J2EE Server

Version 4.0

Contents

Chapter 1. Setting up a Web container in a J2EE server 1
Customizing the Web container in a J2EE server . . 1
Configuring a virtual host 2
Installing Web applications into a J2EE server . . . 3

Chapter 2. Exposing Web applications to HTTP clients 5

Chapter 3. Invoking the Web Installation Verification Program 7

Chapter 4. Configuring HTTP Session Support 9
Configuring session tracking 9
Session security 10
Using cookies for session tracking 11
Using URL rewriting 11
Session clustering 12
Configuring a session cluster 13
Session clustering considerations 14
In-memory session pools 15

Appendix A. webcontainer.conf file . . 17

Appendix B. Migration Considerations 25
Migrating from version 3.5 25
Migrating from V3.02 26
Setting runtime properties 28
Setting Session properties. 28
Accessing services 29
Migrating Web applications to WAR files 29
Servlet reloading 29
Serving servlets by class name 29

Index 31

Chapter 1. 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 via 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. “Chapter 2. Exposing Web applications to HTTP clients” on page 5 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. It’s configuration settings are specified in a webcontainer.conf file provided with the product. You can update the webcontainer.conf file to:

- Configure one or more virtual hosts within a Web container. 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. “Chapter 4. Configuring HTTP Session Support” on page 9 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.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 A. webcontainer.conf file" on page 17 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 A. webcontainer.conf file" on page 17 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. (A tool, such as the IBM WebSphere Studio product, that is used to create JAR files can be used to create a WAR file.)
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.

Chapter 2. 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 25 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/msg/%L%N
```

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.

Chapter 3. Invoking the Web Installation Verification Program

You can use the Web Installation Verification Program (IVP) application, that comes with the product, to verify that your newly configured Web container is functioning properly. This application is contained in the WebSphereSampleApp.ear file which is located in the *applicationserver_root*/bin directory.

Before attempting to run the IVP you must have:

- Defined a J2EE server (see *WebSphere Application Server V4.0 for z/OS and OS/390: Installation and Customization*, GA22-7834)
- Configured the Web server as described in "Chapter 2. Exposing Web applications to HTTP clients" on page 5.

To run the IVP:

1. Customize the Web container in the J2EE server in which you plan to install the Web IVP application to include your host alias(s) (see "Chapter 1. Setting up a Web container in a J2EE server" on page 1).
2. Start the J2EE server. (If the server is already running, you can refresh the server regions associated with the server.)
3. Use FTP to download the WebSphereSampleApp.ear file to the workstation where you plan to run the Systems Management EUI. This file, which resides in the applicationserver_root/bin directory, must be downloaded as a binary file.
4. Use the Systems Management EUI to install the Web IVP application into your J2EE server. (See *WebSphere Application Server V4.0 for z/OS and OS/390: Installation and Customization*, GA22-7834, for details on installing J2EE applications.)
5. After the IVP application is installed, the Web container associated with the J2EE server will be automatically refreshed. At this point, the IVP application will be registered within the J2EE server.
6. Start the Web server. If the initialization process completed successfully, you should receive the following two messages:

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

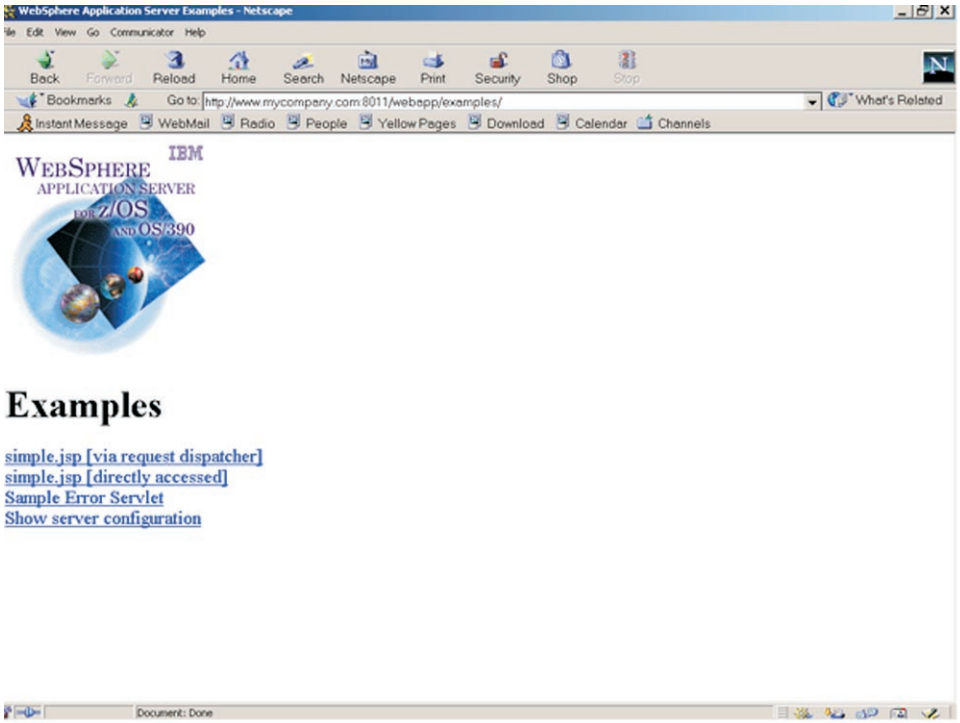
Note: It is possible to get message IMW0235I without the preceding "smiley face" message if the WebSphere for z/OS plugin did not successfully initialize. If you do not receive message IMW0235I, an error occurred during the Web server initialization process.

Using a browser, hit the following url:
http://www.mycompany.com:8011/webapp/examples/index.html If the application is installed successfully you should see the following page:

Then enter the following command from your browser:

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

If the IVP application is successfully installed, you should see the following page:



Chapter 4. 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 A. webcontainer.conf file" on page 17.

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.

	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".**
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.

Appendix A. webcontainer.conf file

Following is a copy of the Web container 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=<name>
#
#        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>
#
#        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=
#
#-------------------------------------------------------------------#
#
#      host.<virtual_hostname>.contextroots=<contextroot>[,<contextroot>]
#
```

```
 This property is used to bind installed Web applications into a
#  specific virtual host. The context root specified here corresponds
#  to the context root bound to the Web application during
#  application deployment.
#
#  One or more context root values can be specified. When specifying
#  multiple context root values, separate each value with a comma.
#  One or more spaces between values are permitted.
#  For example:

#    host.default_host.contextroot=/webapp/examples,  /payroll
#
#  The context root values specified can either be an explicit match
#  to the context root of the deployed Web application or you can
#  use a generic pattern. There are two types of generic patterns:
#
#    1) "/"  which is a catch all context root. All Web application
#       context roots will match this pattern unless there is a
#       more specific context root defined.
#
#       If you are only configuring one virtual host definition in a
#       J2EE Server, you can map all Web applications to that
#       virtual host definition by specifying a context root binding
#       of "/". This specification will allow you to deploy future Web applications
#       into the server without having to update this property.
#       For example:

#           host.default_host.contextroots=/
#
#    2) "/appl/*" which is a generic pattern which means that any
#       Web application context root that begins with "/appl" will
#       match this pattern.
#       For example, the following context roots would all match this pattern:

#           /appl
#           /appl/payroll
#           /appl/hr/benefits
#
#       Note: the use of * is limited to the last position in a
#       context root pattern and must be immediately proceeded by
#       a forward slash.
#
#       The following are examples of valid generic patterns:
#
#           /appl/*
#           /appl/payroll/*
#
#       The following are examples of invalid context root
#       patterns using *:
#
#           *
#           /*
#           /appl*
#           /*/payroll
#
#  The rules for matching a Web application context root to a
#  virtual host context root pattern is as follows:
#
#    1). Find an exact match.
#
#        An exact match will always take precedence over a
#        generic pattern match.
#        For example, if the following context roots are specified for a virtual host:
#
#           host.vh1.contextroots=/webapp/examples/*
#           host.vh2.contextroots=/webapp/examples
#
```

```
#        A Web application with a context root of /webapp/examples
#        will be bound to virtual host name vh2, because it is an
#        exact match.
#
#    2). Find the pattern that most closely matches.
#
#        When multiple generic patterns match a Web application's
#        context root, the generic pattern that matches the most
#        qualifiers of the URI, starting from the left, is
#        considered the best match.
#        For examples, given the following URIs:
#
#           host.vh1.contextroots=/webapp/examples/*
#           host.vh2.contextroots=/webapp/*
#           host.vh3.contextroots=/
#
#        A Web application context root of /webapp/examples/test
#        will be bound to virtual host vh1.
#        A Web application context root of /webapp/test
#        will be bound to virtual host vh2.
#        A Web application context root of /test
#        will be bound to virtual host vh3.
#
#
host.default_host.contextroots=/
#
##-------------------------------------------------------------------#
#
#      host.<virtual_hostname>.mimetypefile=<fully-qualified-filename>
#
#        Specifies the fully qualified filename of the mimetype properties
#        file used for this virtual host.
#
#        The default is:
#        <applicationserver_root>/AppServer/bin/default_mimetype.properties
#
#
host.default_host.mimetypefile=
#
#####################################################################
```

Appendix B. Migration Considerations

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 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 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 for 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 29).

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.
2. You can migrate directly 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 running the V3.5 Application Server in compatibility mode.)

Once you have your V3.5 Application running, you can add a V4.0 Application Server to the same system, provided the V4.0 HFS is mounted at a different mount point than the V3.5 Application Server HFS. Once you have your V4.0 Application Server, "Chapter 2. Exposing Web applications to HTTP clients" on page 5 describes how to indicate to the HTTP server the location of your V3.5 Application Server was.conf file.

You are now set up to continue running your current Web applications on the V3.5 Application Server while developing new applications for the V4.0 Application Server.

When you are ready to migrate your current Web applications to V4.0, or if you want to migrate directly to V4.0, 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 files, using standard Java Archive tools (see "Migrating Web applications to WAR files" on page 29).
3. 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.

Regardless of whether you are migrating directly to V4.0 or migrating 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. In previous SDK versions, such a call would return quietly with no exception.
10. The behavior of **java.net.URL** has changed for cases where a URL instance is constructed from a String. A final slash ('/') is not automatically added to a URL when the URL is constructed without one. For example, the following code:

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

    now results in the following output:

http://www.xxx.yyy

11. The javac complier has a new implementation with the following implications:
    - Inherited members of an enclosing class are now accessible.
    - A local variable or catch clause parameter can be hidden when it is declared within the scope of a like-named method parameter, local variable, or catch clause parameter.
    - It is now illegal for a package to contain a class or interface type and a subpackage with the same name. A package, class, or interface is presumed to exist if there is a corresponding directory, source file, or class file accessible on the classpath or the sourcepath, regardless of its content.
    - A qualified name in a constant expression must be of the form TypeName.identifier.
    - Member classes of interfaces are inherited by implementing classes
12. **java.io.ObjectInputStream** has been optimized to buffer incoming data. This change should improve performance. This change causes ObjectInputStream to more frequently call the multi-byte read(byte[], int, int) method of the underlying stream. If underlying stream classes incorrectly implement this method, serialization failures may occur.
13. A public input method engine SPI as been included so that a client side adapter can be developed and distributed as a separate product and installed into any implementation of the Java 2 platform. Environments that are currently set up to allow text entry using a server-based input method should updated to use a different solution, such as host input methods.

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

http://www.ibm.com/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 V3.x was.conf file session properties can be can be copied 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.

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.

Index

A

adding to a J2EE server 2
alias, associating with a virtual host name 2, 9
Application Server V3.02, migrating from 26
Application Server V3.5, migrating from 25

C

class name, serving servlets by 29
coexistence with V3.5 25
configuring
  a virtual host 2
  adding to a J2EE server 1
  session cluster 13
  session tracking 9
  Web container 1, 11
cookies, not using 11
cookies, using for session tracking 9, 11

D

DB2, using to store session data 13
DB2 table, setting up 13
DNS aliases 2

E

encodeRedirectURL() method 11
encodeURL method 11

G

getSession() method 9

H

host.alias property 2, 17
host.contextroots property 2, 17
host.mimetypefile property 2, 17
host properties 2, 17
HttpSession object 9

I

Installation Verification Program for a Web container 7

J

J2EE server
  adding a Web container to 1
  configuring a virtual host for 2
java.util.Dictionary object 9
javax.servlet.http.HttpServletRequest object 9
javax.servlet.http.HttpSession interface support 9
javax.servlet.http.HttpSessionBindingListener object 9
JDBC, accessing 29
JDNI, accessing 29

M

migrating from V3.02 26
migrating from V3.5 25
migrating Web applications to WAR files 29
migration considerations 25

R

reloading servlets 29
runtime properties 28

S

security for individual sessions 10
serving servlets by class name 29
servlet reloading 29
ServletContextpath, setting prefix associated with 2
servlets, serving by class name 29
session clustering 12
session.cookies.comment property 11, 17
session.cookies.domain property 11, 17
session.cookies.enable property 9, 11, 17
session.cookies.maxage property 11, 17
session.cookies.name property 11, 17
session.cookies.path property 11, 17
session.cookies.secure property 11, 17
session data
  collecting 9
  description of 9
session.dbenable property 13, 17
session.enable property 9, 17
session.invalidationtime property 9, 17
session objects 9
session properties 9, 11, 13, 17, 28
session.protocolswitchrewriting.enable property 11, 17
session security 10
session.tableoverflowenable property 13, 17
session.tablesize property 13, 17
session tracking, configuring 9
session.urlrewriting.enable property 11, 17
sessions
  locking 9
system tools, accessing 29

U

URL rewriting 11
user authentication 10

V

verifying the installation of a Web container 7
virtual host, configuring 2

W

WAR files, migrating Web applications to 29
Web applications, migrating to WAR files 29
Web container
  adding to a J2EE server 1
  configuring 1
  creating 1
Web container Installation Verification Program 7
webcontainer.conf file 17
  properties contained in 2, 9, 11

Program Number: 5655–A98

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.