Servlets -- table of contents

Development

Administration

4.2.1: Developing servlets

Servlets are Java programs that build dynamic client responses, such as Web pages.Servlets receive and respond to requests from Web clients, usually across HTTP, the HyperText Transfer Protocol.

Because servlets are written in Java, they can be ported without modification to different operating systems.Servlets are more efficient than CGI programs because, unlike CGI programs, servlets are loaded into memory once, and each request is handled by a Java virtual machine thread, not an operating system process.Moreover, servlets are scalable, providing support for a multi-application server configuration.Servlets also allow you to cache data, access database information, and share data with other servlets, JSP files and (in some environments) enterprise beans.

Servlet coding fundamentals

In order to create an HTTP servlet, you should extend the `javax.servlet.HttpServlet` class and override any methods that you wish to implement in the servlet. For example, a servlet would override the `doGet` method to handle GET requests from clients.

For more information on the `HttpServlet` class and methods, review articles:

- 4.2.1.3.1: Creating HTTP Servlets
- 4.2.1.3.1.1: Overriding `HttpServlet` methods
- 4.2.1.3.2: Inter-servlet communication

The `doGet` and `doPost` methods take two arguments:

- HttpServletRequest
- HttpServletResponse

The `HttpServletRequest` represents a client's requests. This object gives a servlet access to incoming information such as HTML form data, HTTP request headers, and the like.

The `HttpServletResponse` represents the servlet's response.The servlet uses this object to return data to the client such as HTTP errors (200, 404, and others), response headers (Content-Type, Set-Cookie, and others), and output data by writing to the response's output stream or output writer.

Since `doGet` and `doPost` throw two exceptions (`javax.servlet.ServletException` and `java.io.IOException`), you must include them in the declaration. You must also import classes in the following packages:

Package names	Functions/Objects
`java.io`	PrintWriter
`javax.servlet`	`HttpServlet`
`javax.servlet.http`	`HttpServletRequest` and `HttpServletResponse`

Note: When creating your servlets, do not use the followingreserved words for the class name:

- Description
- Code
- LoadAtStartup
- UserServlet
- DebugMode
- Enabled

Some reserved words such as *UserServlet* can be used in the package names butcreate problems when used as class names.

The beginning of your servlet might look like the following example:

```
import java.io.*;import javax.servlet.*;import javax.servlet.http.*;import java.util.*;public class
MyServlet extends HttpServlet {  public void doGet(HttpServletRequest request,
HttpServletResponse response)      throws ServletException, IOException {
```

After you create your servlet, you must:

1. Compile your servlet using the `javac` command, as for example:
   ```
   javac MyServlet.java
   ```
2. Invoke your servlet using one of the methods described in article:
   4.2.4.4: Providing ways for clients to invoke applications

You can also compile your servlet using the `-classpath` option on the `javac` compiler. To access the classes that were extended, reference the`servlet.jar` file in the `<WAS_root>\lib` directory.Using this method, you issue the following command to compile your servlet:

```
javac -classpath C:\<WAS_root>\lib\servlet.jar MyServlet.java
```

Now that you successfully created, compiled, and tested your servlet on your local machine, you must install it in the WebSphere Application Server runtime. View article 6: Administer applicationsfor this information.

Servlet lifecycle

The javax.servlet.http.HttpServlet class defines methods to:

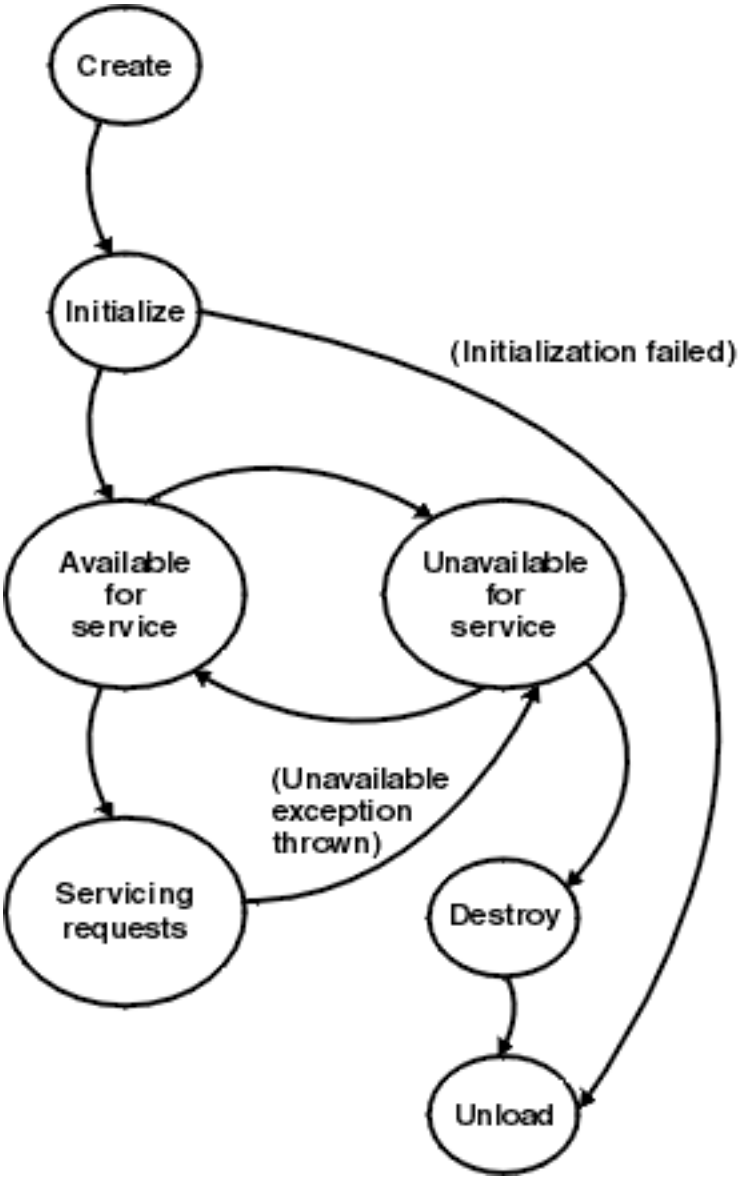
- Initialize a servlet
- Service requests
- Remove a servlet from the server

These are known as life-cycle methods and are called in the following sequence:

1. The servlet is constructed
2. It is initialized with the init method
3. Calls from clients to the service method are handled
4. The servlet is taken out of service
5. It is destroyed with the destroy method
6. The servlet is finalized and the garbage is collected.

Review article 4.2.1.1 for more life cycle information.

4.2.1.1: Servlet lifecycle



Instantiation and initialization

The servlet engine (the Application Server function that processes servlets, JSP files, and other types of server-side include coding) creates an instance of the servlet. The servlet engine creates the servlet configuration object and uses it to pass the servlet initialization parameters to the init method. The initialization parameters persist until the servlet is destroyed and are applied to all invocations of that servlet until the servlet is destroyed.

If the initialization is successful, the servlet is available for service. If the initialization fails, the servlet engine unloads the servlet. The administrator can set an application and its servlets to be unavailable for service. In such cases, the application and servlets remain unavailable until the administrator changes them to available.

Servicing requests

A client request arrives at the Application Server. The servlet engine creates a request object and a response object. The servlet engine invokes the servlet service method, passing the request and response objects.

The service method gets information about the request from the request object, processes the request, and uses methods of the response object to create the client response. The service method can invoke other methods to process the request, such as doGet(), doPost(), or methods you write.

Termination

The servlet engine invokes theservlet's destroy() method when appropriate and unloads the servlet. The Java Virtual Machine performs garbage collection after the destroy.

More on the initialization and termination phases

A servlet engine creates an instance of a servlet at the following times:

- Automatically at the application startup, if that option is configured for the servlet
- At the first client request for the servlet after the application startup
- When the servlet is reloaded

The init method executes only one time during the lifetime of the servlet. It executes when the servlet engine loads the servlet. For the Application Server Version 3, you can configure the servlet to be loaded when the application starts or when a client first accesses the servlet. The init method is not repeated regardless of how many clients access the servlet.

The destroy() method executes only one time during the lifetime of the servlet. That happens when the servlet engine stops the servlet. Typically, servlets are stopped as part of the process of stopping the application.

4.2.1.2: Servlet support and environment in WebSphere

IBM WebSphere Application Server supports the Java ServletAPI from Sun Microsystems. The product builds upon the specificationin two ways.

Article 4.2.1.2.2 describes several IBMextensions to the specification to make it easier to manage sessionstate, create personalized Web pages, generate better servlet errorreports, and access databases.

Article 4.2.1.2.3 describes some complimentary servlets includedwith the product. Add them to Web applications for extended functionality. You can use the WebSphere servlets as theyare, or use them as a basis for creating customized versions.

Beginning with version 3.5.2, WebSphere Application Server added support for the Java ServletAPI 2.2 from Sun Microsystems. See article 4.2.1.2.1afor a description of the Servlet API 2.2 specification.

4.2.1.2.1: Features of Java Servlet API 2.1

Some highlights of the Java Servlet API 2.1 are:

- A request dispatcher wrapper for each resource (servlet)

  A request dispatcher is a wrapper for resources that can process HTTP requests (such as servlets and JSPs) and files related to those resources (such as static HTML and GIFs). The servlet engine generates a single request dispatcher for each servlet or JSP when it is instantiated. The request dispatcher receives client requests and dispatches the request to the resource.

- A servlet context per application

  For the Java Servlet API 2.0, the servlet engine generated a single servlet context that was shared by all servlets. The Servlet API 2.1 provides a single servlet context per application, which facilitates partitioning applications. As explained in the description of the application programming model, applications on the same virtual host can access each other's servlet context.

- Deprecated HTTP session context

  The Servlet API 2.0 HttpSessionContext interface grouped all of the sessions for a Web server into a single session context. Using the session context interface methods, a servlet could get a list of the session IDs for the session context and get the session associated with an ID. As a security safeguard, this interface has been deprecated in the Servlet API 2.1. The interface methods have been redefined to return null.

4.2.1.2.1a: Features of Java Servlet API 2.2

WebSphere Application Server supports Java Servlet API 2.2 and JSP 1.1.

Java Servlet API 2.2 contains many enhancements intended to make servlets part of a complete application framework

These new functions in the Servlet 2.2 specification are ***SUPPORTED*** by WebSphere Application Server:

- response buffering
- WAR files (for depoyment)
- multiple error page support
- welcome file list
- new request mapping logic
- session timeout per Web application
- session scoping per Web application
- MIME mapping table per Web application
  (MIME table now exists at the VirtualHost and Web application)
- request dispatchers by servlet name
- Request dispatchers by relative path
- duplicate header support:
  `(req.getHeaders(name), resp.addHeader())`
- initialization parameters on a Web application
- internationalization improvements:
  `(getLocale(), getLocales())`

The following J2EE extensions in the Servlet 2.2 specification are ***NOT SUPPORTED***:

- J2EE security
- roles
- APIs: `isUserInRole()` and `getUserPrincipal()`
- J2EE-style Form Login
- EJB reference
- resource reference
- environment entry
- reference deployment information in `web.xml`
- security deployment information in `web.xml`
- accessing a JSP file through the URI without the .jsp extension, as for example,
  `../jsp/HitCount`
- creating a sevlet and associating a JSP file as the handler for the servlet

  The Servlet 2.2 specification allows you to associate a JSP tag to the servlet tag. However, the WebSphere Application Server WAR conversion tool does not supportthe `<jsp-file>` tag. The JSP tag association is illustrated in the following code example:

```
<servlet>     <servlet-name>JSPTest</servlet-name>
<jsp-file>/jsp/HitCount.jsp</jsp-file></servlet>/*mapping to URI*/ <servlet-mapping>
<servlet-name>JSPTest</servlet-name>
<url-pattern>/jsp/HitCount.jsp</url-pattern></servlet-mapping>
```

The Servlet 2.2 specification is available atjava.sun.com/products/servlet/index.html

No new classes were added to the Java Servlet API 2.2. specification.The following table provides more information on 27 new methods, 2 new constants and 6 deprecated methods supported by WebSphere Application Server:

New methods	Description
getServletName()	Returns the servlet's registered name
getNamedDispatcher(java.lang.String name)	Returns a dispatcher located by resource name
getInitParameter(java.lang.String name)	Returns the value for the named context parameter
getInitParameterNames()	Returns an enumeration of all the context parameter names
removeAttribute(java.lang.String name)	Added for completeness
getLocale()	Gets the client's most preferred locale

getLocales()	Gets a list of the client's preferred locales as an enumeration of locale objects
isSecure()	Returns **`true`** if the request was made using a secure channel
getRequestDispatcher(java.lang.String name)	Gets a `RequestDispatcher` using what can be a relative path
setBufferSize(int size)	Sets the minimum response buffer size
getBufferSize()	Gets the current response buffer size
reset()	Empties the response buffer, clears the response headers
isCommitted()	Returns true if part of the response has already been sent
flushBuffer()	Flushes and commits the response
setLocale(Locale locale)	Sets the response locale, including headers and charset
getLocale()	Gets the current response locale
UnavailableException(String message)	Replaces `UnavailableException(Servlet servlet, String message)`
UnavailableException(String message, int sec)	Replaces `UnavailableException(int sec, Servlet servlet, String message)`
getHeader(String message)	Returns all the values for a given header, as an enumeration of strings
getContextPath()	Returns the context path of this request
addHeader(String name, String value)	Adds to the response another value for this header name
addDateHeader(String name, long date)	Adds to the response another value for this header name
addIntHeader(String name, int value)	Adds to the response another value for this header name
getAttribute(String name)	`ObjectHttpSession.getValue(String name)`
getAttributeNames()	Replaces `String[] HttpSession.getValueNames()`
setAttribute(String name, Object value)	Replaces `void HttpSession.setValue(String name, Object value)`
removeAttribute(String name)	Replaces `void HttpSession.removeValue(String name)`
New constants	**Description**
SC_REQUESTED_RANGE_NOT_SATISFIABLE	New mnemonic for status code 416
SC_EXPECTATION_FAILED	New mnemonic for status code 417
Newly *deprecated* methods	**Description**
UnavailableException(Servlet servlet, String message)	Replaced by `UnavailableException(String message)`
UnavailableException(int sec, Servlet servlet, String message)	Replaced by `UnavailableException(string message, int sec)`
getValue(String name)	Replaced by `Object HttpSession.getAttribute(String name)`
getValueNames()	Replaced by `numeration HttpSession.getAttributeNames()`
putValue(String message, Object value)	Replaced by`void HttpSession.setAttribute(String name, Object value)`
removeValue(String message)	Replaced by `void HttpSession removeAttribute(String name)`

4.2.1.2.2: IBM extensions to the Servlet API

The Application Server includes its own packages that extend and add to the Java Servlet API. Those extensions and additions make it easier to manage session state, create personalized Web pages, generate better servlet error reports, and access databases. The Javadoc for the Application Server APIs is installed in the product *product_installation_root*\web\apidocs directory.

The Application Server API packages and classes are:

- `com.ibm.servlet.personalization.sessiontracking` package

  This Application Server extension to the Java Servlet API records the referral page that led a visitor to your Web site, tracks the visitor's position within the site, and associates user identification with the session. IBM has also added session clustering support to the API.

- `com.ibm.websphere.servlet.session.IBMSession` interface

  Extends HttpSession for session support and increased Web administrators' control in a session cluster environment.

- `com.ibm.servlet.personalization.userprofile` package

  Provides an interface for maintaining detailed information about your Web visitors and incorporate it in your Web applications, so that you can provide a personalized user experience. This information is made persistent by storing it in a database.

- `com.ibm.websphere.userprofile` package

  User profile enhancements

- `com.ibm.db` package

  Includes classes to simplify access to relational databases and provide enhanced access functions (such as result caching, update through the cache, and query parameter support).

- `com.ibm.websphere.servlet.error.ServletErrorReport` class

  A class that enables the application to provide more detailed and tailored messages to the client when errors occur. See the enhanced servlet error reporting article, 4.2.1.3.5, for details.

- `com.ibm.websphere.servlet.event` package

  Provides listener interfaces for notifications of application lifecycle events, servlet lifecycle events, and servlet errors. The package also includes an interface for registering listeners. See the package Javadoc for details.

- `com.ibm.websphere.servlet.filter` package

  Provides classes that support servlet chaining. The package includes the ChainerServlet, the ServletChain object, and the ChainResponse object. See the servlet filtering article, 4.2.1.3.4, for more details.

- `com.ibm.websphere.servlet.request` package

  Provides an abstract class, HttpServletRequestProxy, for overloading the servlet engine's HttpServletRequest object. The overloaded request object is forwarded to another servlet for processing. The package also includes the ServletInputStreamAdapter class for converting an InputStream into a ServletInputStream and proxying all method calls to the underlying InputStream. See the Javadoc for details and examples.

- `com.ibm.websphere.servlet.response` package

Provides an abstract class, HttpServletResponseProxy, for overloading the servlet engine's HttpServletResponse object. The overloaded response object is forwarded to another servlet for processing. The package includes the ServletOutputStreamAdapter class for converting an OutputStream into a ServletOutputStream and proxying all method calls to the underlying OutputStream. The package also includes the StoredResponse object that is useful for caching a servlet response that contains data that is not expected to change for a period of time, for example, a weather forecast. See the Javadoc for details and examples.

4.2.1.2.3: Using the WebSphere servlets for a head start

IBM Application Server provides internal (built-in) WebSphere servlets that you can add to your Web applications to enable optional functions.

The tables below describe each WebSphere servlet and how to use the Java console to add it to a Web application. To determine whether a WebSphere servlet currently belongs to a Web application, check the Web application configuration for the presence of the servlet by its administrative name.

Invoke servlets by class name

Objective	Invoke servlets by class or code names (such as MyServletClass)
Servlet administrative name	invoker
Servlet code	com.ibm.servlet.engine.webapp.Invoker
How to add to Web application	When using the Console -> Task -> Configure a Web application wizard, specify to serve servlets by classname. For an existing Web application, use the Console -> Tasks -> Add a servletwizard.
More information	Using the Invoker servlet is considered a security exposure that can be avoided by performingcertain administrative tasks. See the Related information for details. The default invoker's URL in Servlet 2.2 ***compliance*** modeis `/servlet/*` , not `/servlet/`.See file, New Servlet Engine option for migrating applications to Servlet 2.2, for information on the two modes: compliance versus compatibility.

Serve files without specifically configuring them

Objective	Serve HTML, servlets, or other files in the Web application document root without extra configuration steps. For HTML files, you will not need to add a pass rule to the Web server. For servlets, you will not need to explicitlyocnfigure the servlets in the WebSphere administrative domain.
Servlet administrative name	file
Servlet code	com.ibm.servlet.engine.webapp.SimpleFileServlet
How to add to Web application	When using the Console -> Task -> Configure a Web application wizard, specifyto enable the file servlet. For an existing Web application, use the Console -> Tasks -> Add a servletwizard.
More information	This servlet handles files in the application document root whose URLs are not covered by the configured servlet URLs

Enable Web applications to serve JSP files

Objective	Enable the JSP page compiler to allow Web application to handle JSP files
Servlet administrative name	See section 4.2.1.2
Servlet code	See section 4.2.1.2
How to add to Web application	When using the Console -> Task -> Configure a Web application wizard, specify a JSP level for the Web application. For an existing Web application, use the Console -> Tasks -> Add a JSP enabler wizard.
More information	Adding a JSP processor to an application is required if the Web application contains JSP files. • 4.2.1.2: JSP processors • 6.6.10: Administering JSP files

Enable an error page without having to write one

Objective	Enable error reporting through an error page, withoutwriting your own error page
Servlet administrative name	ErrorReporter
Servlet code	com.ibm.servlet.engine.webapp.DefaultErrorReporter
How to add to Web application	Configure the Web application, then add the ErrorReporter servlet byusing the Console -> Tasks -> Add a servlet wizard.
More information	4.2.1.3.5: Enhancing servlet error reporting

Enable servlet chaining

Objective	Enable a servlet chain, in which servlets forward output and responses to other servlets for processing
Servlet administrative name	Chainer
Servlet code	com.ibm.websphere.servlet.filter.ChainerServlet
How to add to Web application	Configure the Web application, then add the Chainer servlet byusing the Console -> Tasks -> Add a servlet wizard.
More information	• 4.2.1.3.4: Filtering and chaining servlets

4.2.1.2.3.1: Avoiding the security risks of invoking servlets by class name

Anyone enabling the Invoker servlet to serve servlets by their class names

Anyone enabling the "serve files by class name" function in WebSphere Application Server, should take steps to avoid potential security risks. The administrator should remain aware of each and every servlet classplaced in the classpath of an application, even if the servlets are to be invoked by their classnames.

Appending `/$/foo` to the URL allows you to access the servlet URL, butthe URL is then misunderstood by the runtime environment.This type of URL may create a security exposure.The best practice for securing servlets is to follow the Java security specifications documented in the Securing applications section.

A Web site may inadvertently include malicious HTML tags or scripts in a dynamically generated page based on unvalidated input from untrustworthy sources.By accessing a malicious URL and then accessing an application server, a usermay unknowingly execute script code on his machine that exposes the data received from the server. The browser executes the script on the user machine without the knowledge of the user.

The malicious tagsthat can be embedded in this way are <SCRIPT> and </SCRIPT>.

This problem can be prevented if the server generated pages are encoded to prevent thescripts from executing.Developers generating responses containing client data, based on servlet or JSP requests, canencode the responses using the following method:

```
com.ibm.websphere.servlet.response.ResponseUtils.encodeDataString(String)
```

Visit the Cert advisories Web sitefor more information.

Protecting servlets

To protect each servlet, the administrator needs to:

1. Configure a Web resource based on the servlet class name, such as:

   ```
   /servlet/SnoopServlet
   ```

   for SnoopServlet.class
2. Add the Web resource to the Web Path list of the Invoker servlet in the Web applicationto which the servlet belongs.
3. Use the Configure Resource Security wizard in the Java administrative console to secure the Web resource.

Also, the administrator needs to secure the Invoker servlet itself.

Details

WebSphere security is based on defining, and then securing, URIs (knownas Web resources) for servlets. This allows an administrator to applydifferent security levels to different paths for accessing the same servlet.Also, Web resources are logical designations that are not guaranteed to match servlet class names. For these reasons, actual class names are irrelevant to WebSphere security unless you explicitly specify that you want to protect the path for invoking a servlet by its class name.

When a Web application allows users to invoke servlets by class name, the administrator is able to drop servlets into a Web application without havingto explicitly define them in WebSphere systems administration.

Suppose that the WebSphere administrator drops in a servlet class tobe invoked by its class name. Even if a servlet corresponding to the same class name is defined and protected, users will be able to invoke the servlet by class name without any security checks. (The exception is if the administrator has created a Web resource corresponding to the servletclass name, as described in the above steps).

Undefined servlets remain unprotected unless steps are taken to assign secure Web resources to them based on their class names.

4.2.1.2.3b: Security risk example of invoking servlets by class name

Anyone enabling the "serve files by class name" function in WebSphere Application Server, should take steps to avoid potential security risks. The administrator should remain aware of each and every servlet classplaced in the classpath of an application, even if the servlets are to be invoked by their classnames.

 A Web site may inadvertently include malicious HTML tags or scripts in a dynamically generated page based on unvalidated input from untrustworthy sources.By accessing a malicious URL and then accessing an application server, a usermay unknowingly execute script code on his machine that has full access to the data and resources on that machine. The browser executes the script on the user machine without the knowledge of the user.

The malicious tagsthat can be embedded in this way are <SCRIPT> and </SCRIPT>.

This problem can be prevented if the server generated pages are encoded to prevent thescripts from executing.Developers generating responses containing client data, based on servlet or JSP requests, canencode the response data using the following method:

```
com.ibm.websphere.servlet.response.ResponseUtils.encodeDataString(String)
```

Visit the Cert advisories Web sitefor more information.

4.2.1.3: Servlet content, examples, and samples

Click the related topics to focus on particular aspects of servletdevelopment, including example and sample code.

4.2.1.3.1: Creating HTTP servlets

To create an HTTP servlet, as illustrated in ServletSample.java:

1. Extend the HttpServlet abstract class.
2. Override the appropriate methods. The ServletSample overrides the doGet() method.
3. Get HTTP request information, if any.

   Use the HttpServletRequest object to retrieve data submitted through HTML forms or as query strings on a URL. The ServletSample example receives an optional parameter (myname) that can be passed to the servlet as query parameters on the invoking URL. An example is:

   `http://your.server.name/application_URI/ServletSample?myname=Ann`

   The HttpServletRequest object has specific methods to retrieve information provided by the client:

   - getParameterNames()
   - getParameter(java.lang.String name)
   - getParameterValues(java.lang.String name)
4. Generate the HTTP response.

   Use the HttpServletResponse object to generate the client response. Its methods allow you to set the response headers and the response body. The HttpServletResponse object also has the getWriter() method to obtain a PrintWriter object for sending data to the client. Use the print() and println() methods of the PrintWriter object to write the servlet response back to the client.

4.2.1.3.1.1: Overriding HttpServlet methods

HTTP servlets are specialized servlets that can receive HTTP client requests and return a response. To create an HTTP servlet, subclass the HttpServlet class. A servlet can be invoked by its URL, from a JavaServer Page (JSP), or from another servlet.

Methods to override

The `javax.servlet.http.HttpServlet` class contains the init, destroy, and service methods. The init and destroy methods are inherited, while the service methodimplementation is specific to HttpServlet. The method behaviors are described below; however, you might want to override methods in order to provide specialized behavior in your servlet.

- **init**

  The default init method is usually adequate but can be overridden with a custom init method, typically to register application-wide resources. For example, you might write a custom init method to load GIF images only one time, improving the performance of servlets that return GIF images and have multiple client requests. Other examples are initializing a database connection and registering servlet context attributes.

  The Java Servlet API 2.1 provides a new init method: `init()`, the no argument init method that is inherited from the superclass GenericServlet. The GenericServlet also implements the ServletConfig object. The benefit is that when you use the no-argument init method in your servlet, you do not need to call `super.init(config)`. The reason is that servlet engines that implement the Servlet API 2.1 call the servlet's `init(ServletConfig config)` method behind the scenes. In turn, the GenericServlet calls the servlet's init() method.

  If a servlet exception is thrown inside the init method, the servlet engine will unload the servlet. The init method is guaranteed to complete before the service method is called.

- **destroy**

  The default destroy method is usually adequate, but can be overridden.Override the destroy method if you need to perform actions during shutdown. For example, if a servlet accumulates statistics while it is running, you might write a destroy() method that saves the statistics to a file when the servlet is unloaded. Other examples are closing a database connection and freeing resources created during the initialization.

  When the server unloads a servlet, the destroy method is called after all service method calls complete or after a specified time interval. Where threads have been spawned from within service method and the threads have long-running operations, those threads may be outstanding when the destroy method is called. Because this is undesirable, make sure those threads are ended or completed when the destroy method is called.

- **service**

  The service method is the heart of the servlet. Unlike the init and destroy methods, it is invoked for each client request. In the HttpServlet class, the service method already exists. The default service function invokes the doXXX method corresponding to the method of the HTTP request. For example, if the HTTP request method is GET, doGet method is called by default. Because the HttpServlet.service method checks the HTTP request method and calls the appropriate handler method, it is usually not desirable to override the service method. Rather, override the appropriate doXXX methods that the servlet supports.

4.2.1.3.2: Inter-servlet communication

There are three types of servlet communication:

- Accessing data within a servlet's scope
- Forwarding a request and including a response from another servlet using the RequestDispatcher
- Application-to-application communication via the ServletContext

Sharing data within scope

JavaServerPages (JSPs) use this method to share data through beans. The ability of servlets to share data depends on the scope of the bean. The possible scopes are request, session, and application.

Forwarding and including data

For session-scoped data and attributes, use the HttpSession.setAttribute and getAttribute methods to set and get attributes in the HttpSession object. Session-scoped beans and objects bound to a session are examples of session-scoped objects.

For the Servlet API 2.1, an HttpSession object is only accessible to the Web applications and servlets that are a part of that session. In the Servlet API 2.1, a servlet cannot determine the ID of another session and request its SessionContext, because the HttpSessionContext and related methods are deprecated (returns null).

For application-scoped data, use the RequestDispatcher's forward and include methods to share data among applications. The forward method sends the HTTP request from one servlet to a second servlet for additional processing. The calling servlet adds the URL and request parameters in its HTTP request to the request object passed to the target servlet. The forwarding servlet must not have committed any output to the client. The target servlet generates the response and returns it to the client.

The include method enables a receiving servlet to include another servlet's response data in its response. The included servlet cannot set response headers. The receiving servlet can fully access the request object but can only write data to the ServletOutputStream or PrintWriter of the response object. If the servlets use session tracking, you must create the session outside of the included servlet. The RequestDispatcher.forward method is similar in function to the HttpServiceResponse.callPage method previously supported for JSP development.

Application-to-application communication

Web applications share data through the ServletContext. A Web application has a single servlet context. A ServletContext object is accessible to any Web application associated with a virtual host. Servlet A in application A can obtain the ServletContext for application B in the same virtual host. After Servlet A obtains the servlet context for B, it can access the request dispatcher for servlets in application B and call the getAttribute and setAttribute methods of the servlet context. An example of the coding in Servlet A is:

```
appBcontext = appAcontext.getContext("/appB");
appBcontext.getRequestDispatcher("/servlet5");
```

4.2.1.3.2.1: Forwarding and including data (request and response)

When the servlet engine calls the service method of an HTTP servlet, it passes two objects as parameters:

- HttpServletRequest (the Request object)
- HttpServletResponse (the Response object)

The servlet communicates with the server andultimately with the client through these objects. The servlet reads the Request object from a ServletInputStream. The servlet can invoke the Requestobject's methods to get information about the client environment, theserver environment, and any information provided by the client (for example,form information set by GET or POST).The servlet invokes the Response object's setter methods to return the client response. However, if the servlet is part of a servlet chain, it might pass its response object to another servlet for further processing.

4.2.1.3.2.2: Example: Servlet communication by forwarding

In this example, the forward method is used to send a message to a JSP file (a servlet) that prints the message. The forwarding servlet code is:

```
import java.io.*;import javax.servlet.*;import javax.servlet.http.*;public class UpdateJSPTest extends HttpServlet{   public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException     {        String message = "This is a test";  req.setAttribute("message", message);        RequestDispatcher rd = getServletContext().getRequestDispatcher("/Update.jsp");        rd.forward(req, res);      }}
```

The JSP file is:

```
<html><head></head><body><h1><servlet code=UpdateJSPTest></servlet></h1><%    String message = (String) request.getAttribute("message");    out.print("message: <b>" + message + "</b>");%><p><ul><% for (int i = 0; i < 5; i++)        {       out.println ("<li>" + i);  }%></ul></body></html>
```

4.2.1.3.3: Using page lists to avoid hard coding URLs

IBM WebSphere Application Server supports page lists, which allow applicationdevelopers to prevent hard-coding URLs in servlets and JSP files. To learn howpage lists work, and their advantages, see the page lists descriptioncited in the Related information below.

Use IBM WebSphere Studio to develop (1) servlets that support page lists, and(2) their accompanying .servlet configuration files that specify the page lists.Alternatively, use materials supplied by IBM WebSphere Application ServerVersion 3.x to manually create the two items.

Regardless of how you obtain them, servlets and their .servlet configurationfiles can be deployed in an IBM WebSphere Application Server environment.

See the Related information for instructions for using .servletconfiguration files obtained from either Studio or WebSphere ApplicationServer.

4.2.1.3.3.1: Obtaining and using servlet XML configuration files (.servlet files)

The IBM WebSphere Studio provides wizards that generate servlets withaccompanying XML servlet configuration files (.servlet files).

If you do not have access to the Studio, you can manually implementXML servlet configuration files. The servlet must also be modified orcreated to support the use of a .servlet file for its configuration.

Using .servlet files from IBM WebSphere Studio

1. Use IBM WebSphere Studio to create a servlet and .servlet files. See theStudio documentation for instructions.
2. Deploy the compiled servlet and its XML servlet configuration file on the applicationserver.

Using manually configured .servlet files

1. Create or obtain a servlet that extends the PageListServlet class.
2. Use the XMLServletConfig class to create an XML servlet configuration file for the servlet instance.
3. Deploy the compiled servlet and its XML servlet configuration file.

4.2.1.3.3.1.1: Extending PageListServlet

IBM WebSphere Application Server supplies the PageListServlet, the superclass of servlets that load pages contained in the page list element (<page-list>) of an XML servlet configuration file.

Implement a servlet that supports the use of XML configuration files(.servlet files) and page lists by extending the PageListServlet class.

The PageListServlet has a callPage() method that invokes a JavaServer Page in response to an HTTP request for a page in the page list.

The PageListServlet.callPage() method receives as input:

- A page name from the page-list element of the XML configuration file
- The HttpServletRequest object
- The HttpServletResponse object

In structuring the servlet code, keep in mind that the PageListServlet.callPage() method is not an exit. Any servlet code that follows the callPage() method invocation will be run after the invocation.

See the Related information for an example of a servlet that extends thePageListServlet.

4.2.1.3.3.1.1.1: Example: Extending PageListServlet

SimplePageListServlet is an example of a servlet that extends the PageListServlet class and uses its callPage() method to invoke a JSP:

```
public class SimplePageListServlet extends com.ibm.servlet.PageListServlet {   public void
service(HttpServletRequest req, HttpServletResponse resp)   throws ServletException, IOException   {
try{         setRequestAttribute("testVar", "test value", req);
setRequestAttribute("otherVar", "other value", req);         String pageName =
getPageNameFromRequest(req);         callPage(pageName, req, resp);      }      catch(Exception e){
handleError(req, resp, e);      }   }}
```

4.2.1.3.3.1.2: Using XMLServletConfig to create .servlet configuration files

IBM WebSphere Application Server supplies the XMLServletConfig class for creating XML servlet configuration files (*servlet_instance_name*.servlet files).

Write a Java program that uses the XMLServletConfig class to generate a servlet configuration file. The XMLServletConfig class provides methods for setting and getting the file elements and their contents.

See the comments in the XMLServletConfig class for an explanation of how to use it.

4.2.1.3.3.1.3: XML servlet configuration file syntax (.servlet syntax)

Each XML configuration file must be a well-formed XML document. The files are not validated against a Document Type Definition (DTD). This article describes the syntax, as illustrated by theexample cited in Related information.

For the Application Server to use an XML servlet configuration file to load a servlet instance, the file must contain at least the code element. For a PageListServlet, the XML configuration file must contain at least the code element and the page-list element.

Although there is no DTD, it is recommended that all elements appear in the order shown in the example. The elements (also known as *tags*) are:

Tag	Description
servlet	The root element of an XML configuration file. The XMLServletConfig class automatically generates this element.
code	The class name of the servlet (without the .class extension), even if the servlet is in a JAR file
description	A user-defined description of the servlet
init-parameter	The attributes of this element specify a name-value pair to be used as an initialization parameter. A servlet can have multiple initialization parameters, each within its own init-parameter element.
page-list	Identifies the JavaServer pages to be called depending on the path information in the HTTP request. The page-list element can contain the following child elements: ● default-page: Contains a uri element that specifies the location of the page to be loaded, if the HTTP request does not contain path information ● error-page: Contains a uri element that specifies the location of the page to be loaded, if the handleError() method sets the request attribute "error" ● page: Contains a uri element that specifies the location of the page to be loaded if the HTTP request contains the page name. A page-list element can contain multiple page elements.

4.2.1.3.3.1.4: Example: XML servlet configuration file

```
<?xml version="1.0"?><servlet>  <code>SimplePageListServlet</code>  <description>Shows how to use
PageListServlet class</description>  <init-parameter name="name1" value="value2"/>  <page-list>
<default-page>      <uri>/index.jsp</uri>    </default-page>    <error-page>
<uri>/error.jsp</uri>    </error-page>    <page>      <uri>/TemplateA.jsp</uri>
<page-name>page1</page-name>    </page>    <page>      <uri>templateB.jsp</uri>
<page-name>page2</page-name>    </page>  </page-list></servlet>
```

4.2.1.3.4: Filtering and chaining servlets

The Application Server supports two kinds of filtering:

- *MIME-based filtering* involves configuring the servlet engine to forward HTTP responses with the specified MIME type to the designated servlet for further processing.
- Servlet chaining involves defining a list (a sequence) of two or more servlets such that the request object and the ServletOutputStream of the first servlet is passed to the next servlet in the sequence. This process is repeated at each servlet in the list until the last servlet returns the response to the client.

4.2.1.3.4.1: Servlet filtering with MIME types

To configure MIME filters, use an administrative client to configure recognized MIME types for virtualhosts containing servlets.

4.2.1.3.4.2: Servlet filtering with servlet chains

To configure a servlet chain, use the administrative console to:

- Define the following initialization parameter and value for the ChainerServlet:

Parameter	Value
chainer.pathlist	*/first_servlet_URL /next_servlet_URL*

The chainer.pathlist is a space-delimited list of servlet URLs. For example, if you want the sequence of servlets to be three servlets that you added to the examples application (servletA, servletB, servletC), specify:

Parameter	Value
chainer.pathlist	/servletA /servletB /servletC

- To invoke a servlet chain, invoke the servlet URL of the ChainerServlet in your application. instance, for example, http://*your.server.name*/webapp/example/abc.

4.2.1.3.5: Enhancing servlet error reporting

A servlet can report errors by:

- Calling the ServletResponse.sendError method
- Throwing an uncaught exception within its service method

The enhanced servlet error reporting function in IBM WebSphere Application Server provides an easier way to implement error reporting. The error page (a JSP file or servlet) is configured for the application and used by all of the servlets in that application. The new mechanism handles caught and uncaught errors.

To return the error page to the client, the servlet engine:

1. Gets the ServletContext.RequestDispatcher for the URI configured for the application error path.
2. Creates an instance of the error bean (type `com.ibm.websphere.servlet.errorServletErrorReport`). The bean scope is request, so that the target servlet (the servlet that encountered the error) can access the detailed error information.

For the Application Server, the ServletResponse.sendError() method has been overriden to provide the functionality previously described. The overriden method is shown below:

```
public void sendError(int statusCode, String message){   ServletException e = new
ServletErrorReport(statusCode, message);   request.setAttribute(ServletErrorReport.ATTRIBUTE_NAME,
e);   servletContext.getRequestDispatcher(getErrorPath()).forward(request, response);}
```

4.2.1.3.5.1: Public methods of the ServletErrorReport class

To create an error JSP or servlet, you need to know the public methods of the `com.ibm.websphere.servlet.error.ServletErrorReport` class (the error bean), which are:

```
public class ServletErrorReport extends ServletException{       //Get the stacktrace of the error as a string   public String getStackTrace()   //Get the message associated with the error.    //The same message is sent to the sendError() method.   public String getMessage()      //Get the error code associated with the error. //he same error code is sent to the sendError() method. //This will also be the same as the status code of the response.        public int getErrorCode()       //Get the name of the servlet that reported the error   public String getTargetServletName()}
```

4.2.1.3.5.2: Example: JSP file for handling application errors

As illustrated in the following code example, specify "ErrorReport" for the id value.The error page loads an instance of code from the request space named "ErrorReport"to read the properties. If the default scope (scope="page") is used, a new instance of the code is created and the properties are blank.

```
<html><jsp:useBean id="ErrorReport"
class="com.ibm.websphere.servlet.error.ServletErrorReport"scope="request" /><head>   <title>
ERROR: <%= ErrorReport.getErrorCode() %>   </title></head><body><H1>  This error occured while
processing the servlet named:   <%= ErrorReport.getTargetServletName() %></H1><B>My Message: </B><%=
ErrorReport.getMessage() %><BR><BR><B>My StackTrace: </B><%= ErrorReport.getStackTrace()
%><BR></body></html>
```

If you do not want to write your own error, consider adding the optional internal servlet, com.ibm.servlet.engine.webapp.DefaultErrorReporter, to your Web application.

4.2.1.3.6: Serving servlets by classname

IBM WebSphere Application Server provides a WebSphere servletthat you can add to your Web applications. Web applications thatcontain the servlet can serve servlets by theservlet classnames (such as MyServletClass). No additional stepsare required.

See the details and instructions.

4.2.1.3.7: Serving all files from application servers

IBM WebSphere Application Server provides a WebSphere servletthat you can add to your Web applications. Web applications thatcontain the servlet can serve HTML, eliminating the need to adda pass rule to the Web server for serving the same HTML files. No additional steps are required.

See the details and instructions.

4.2.1.3.8: Obtaining the Web application classpath from within a servlet

To have a servlet or JSP-generated servlet detect the classpathof the Web application to which it belongs, get the

`com.ibm.websphere.servlet.application.classpath`

attribute from the ServletContext.

6.6.7: Administering servlet engines (overview)

A servlet engine configuration provides information about the applicationserver component that handles servlet requests forwarded bythe Web server. The administratorspecifies servlet engine properties including:

- Application server on which the servlet engine runs
- Number and type of connections between the Web server and servlet engine
- Port on which the servlet engine listens

6.6.7.0: Servlet engine properties

Application Server

Specifies the application server with which to associate the servlet engine.

Current State

Indicates the state the servlet engine is currently in. The next time the servlet engine is started, it will try to change to its desired state setting.

Desired state

Indicates the state the servlet engine should have the next time it is started.

Max Connections

Specifies the maximum number of concurrent resource requests to allow.

Max Connections in use

Specifies the Max Connections value currently in use.

Port

Specifies the port the servlet engine will listen on for servlet requests from the Web server.

Port in use

Specifies Port value currently in use.

Queue Type (Transport Type)

Specifies the connectivity type for communication between Web servers and application servers to obtain servlet requests:

OSE	For routing requests locally.
HTTP	Not recommended at this time
None	

If you specify OSE, specify these properties related to Queue Type.

Clone Index

Specifies a unique numerical identifier for this servlet engine instance.

Native Log File

Specifies the log file that will be considered "standard out" for tracingand debugging of the native code of the product. Specify either:

- A file name, with product_installation_root/logs assumed to be the directory
- A fully qualified path to a log file

Queue Name

Specifies the name of the queue for holding requests tobe processed by the servlet engine.

Select Log File Mask

Specifies one or more levels of messages to log -- error, warning, informational, or trace.

Transport Type

Specifies the communication protocol type to use with the OSE transport:

- Local pipes
- INET sockets
- JAVA TCP/IP (not currently supported)

See the servlet engine tuning section of the Tuning Guide for suggested values, typically based on the operating system.

Queue Type in use

Indicates the queue type currently in use (see Queue Type description above).

Servlet Engine Mode FP 3.5.2

Specifies how servlets will be supported (in terms of how the specification levels are enforced). Consider the implications carefully before changing this setting. See article 3.3.2a for a discussion and details.

If you switch the servlet engine to full compliance mode, adjust the Servlet Web Path Lists of servlets running in this servlet engine, to keep your Web applications from breaking. Add a /* to the end of each Web path.

For example, if the path for a servlet is:

```
default_host/WebSphereSamples/servlet
```

then change it to:

```
default_host/WebSphereSamples/servlet/*
```

Servlet Engine Name

Specifies a servlet engine name. The name must be unique in the scope of theapplication server. In other words, you can create two servlet engines with the samename as long as each servlet engine is associated with a different applicationserver.

Start Time

Indicates the time at which the servlet engine was most recently started. A valueof "--" indicates the servlet engine has not been started since the administrativeserver started.

6.6.7.1: Administering servlet engines with the Java administrative console

This article extends article 6.6.7 (the overview of administering servlet engines) with information specific to the Java console.

The table answers the most basic questions. See the Related informationfor links to detailed instructions and resource properties.

Does the console provide full functionality for administering this resource?	Yes
How is this resource represented in the console tree views?	The Type tree contains a Servlet Engines folder object. The Topology tree can contain zero or more existing servlet engines. Their names vary;they are supplied by the administrator. Use the View menu on the console menu bar to toggle between tree views.
Any task wizards for manipulating this resource?	On the console menu bar: Console -> Task -> Create a servlet engine

6.6.7.1.1: Configuring new servlet engines with the Java administrative console

The product offers several ways to configure new servlet engines:

- By clicking Console -> Tasks -> Create a servlet engine from theconsole menu bar.
- By clicking Create a servlet engine from the drop-down list on theWizards toolbar button.
- Using menus on resources in the Topology and Type trees(see Related information)

The first two methods lead to the Create a servlet enginetask wizard, for which detailed help is provided here.

1. Follow the wizard instructions.
    - Specify a name by which to manage the servlet engine.
    - Specify the application server to contain the servlet engine.

    Click Next to proceed.
2. Specify servlet engine properties.
3. Click Finish.

6.6.7.3: Administering servlet engines with the Web console

Use the Web console to edit the configurations of servlet engines, which are responsible for provided needed servicesto running Web modules and their contained servlets and JSP files. Each application server runtime has one logical servlet engine, which you can modify butnot create or remove.

Work with objects of this type by locating them in the tree on the left side of the console:

Click **Tasks** -> **Create Objects** -> **Create Servlet Engine**

When creating a servlet engine, you must specifyan existing application server to contain it. Existing servlet engines and application servers in the administrative domain are displayed in the **Resources** section of the navigation tree.

 Creations and changes made with this console are not appliedto the administrative domain until you Commit them. Refer to section 6.6.0.3.5 for details.

6.6.7.4: Property files pertaining to servlet engines

The servlet engine properties are in file:

- *servlet_engine.properties*

This file is located in directory:

`<WebSphere/Appserver>/properties`

6.6.8: Administering Web applications (overview)

The servlets and JavaServer Pages (JSP) files in a Web application share a servletcontext, meaning they share data and information about the execution environment,including a Web application classpath.

Approaches to configuring Web applications

There are two basic approaches to configuring Web applications. Theadministrator can configure a Web application:

- From the bottom up
- From the top down

To configure a Web application from the **bottom up**, the administrator can first explicitlyconfigure the individual servlets that will eventually comprise the Web applications.When configuring a servlet, the administrator specifiesthe name and location of the servlet class file, and other information necessaryfor enabling the administrator to manage the servlet.

The administrator can combine one or more explicitly-defined servlets and Web resources into an Webapplication, allowing them to be managed as a logical unit (the Web application).

Because they are explicitly configured, the servlets can also be managedindividually. For example, the administrator can unload a servlet from the Web application without causingthe rest of the application to become unavailable to users.

The administrator can also configure a Web application from the **top down**. Thistechnique might be familiar to an administrator who has used Web serverproducts or IBM WebSphere Application Server Version 2.

Instead ofexplicitly defining each component (servlet, Web page, and so on), the administratorspecifies the directories in which he or she plans to place the components of each type.

In the simplest case, each Web application has one directory for servlets and anotherfor Web resources. Any servlet placed in the designated servlet directory becomes partof the Web application, and similarly any Web pages and JavaServer Pages (JSP) files arepicked up from the designated Web resource directory.

Because the servlets are not explicitly defined, they cannot be managed or monitoredindividually.

Web applications inside enterprise applications

A Web application can be part of an enterprise application (an "application" forshort). In the simplest case, an enterprise application is simply a "wrapper" for aWeb application -- the files that comprise the application are exactly the samefiles that comprise the *Web* application.

In such a case, why bother to add a Web application to an enterprise application?An enterprise application help file discusses the benefits.

In a more complex case, an application might contain multiple Web applicationsand (in the case of IBM WebSphere Application Server *Advanced Edition*) some enterprise beans as well.

Configuring Web applications directly in WebSphere systems administration

The administrator should understand a few main settings as he or she configuresWeb applications:

- Classpath

  Specifies where to find the servlets that belong to the application.

  The classpath can specify a *directory* containing servlets, or can specify each servlet explicitly.

  It can also specify the location of other files supporting the Web application.
- Document root

  Specifies where to find the Web pages and JSP files belonging to the Web application.
- Web path

  Combined with the virtual host, specifies what users will type in a Web browser to access the Web application.

The administrator can also specify properties such as:

- Servlet filtering parameters
- Affiliation with a virtual host
- Whether to reload servlets whose class files have changed
- Whether to temporarily suspend the Web application from use
- Servlet context attributes

Classpath considerations

An important classpath-related setting to note is the Module Visibility. This application server setting impacts the portability of applications and standalone modules from other WebSphere Application Server versions and editions. If your existing module does not run as-is when you transfer it to Version 4.0, you might need to reassemble an existing module or change the module visibilitysetting.

See the information on setting classpaths for a full discussion of classpath considerations. See the applicationserver property reference for information about the module visibility setting.

Identifying a welcome page for the Web application

The default welcome page for your Web application is assumed to be named index.html. For example, if you have an application with a Web path of:

```
/webapp/myapp
```

then the default page named index.html can be implicitly accessed using the following URL:

```
http://hostname/webapp/myapp
```

FP 3.5.2 Version 3.5.2 introduces a Welcome Files setting, as described by the Servlet 2.2 specification. See the Web application properties for details.

FP 3.5.2 Converting WAR files

Version 3.5.2 (Fix Pack 2 applied to Version 3.5 base) introduces a new wayto introduce Web applications into the WebSphere environment. The productnow consumes and converts WAR files into WebSphere configurations.

Alternatively, you can continue to configure Web applications directly in WebSphere Application Server systems administration. The latter allows you to add WebSphereservlets to your Web applications to extend their functionality.

You can use either the Java console (WebSphere Administrative Console) or command line programs to convert WAR files.

Utilizing servlets available from WebSphere

See section 4.2.1.2.3 for information about addingcomplimentary WebSphere servlets to Web applicationsto provide functions such as JSP enablement, errorreporting, file serving, and the ability to invokeservlets by classname.

6.6.8.0: Web application properties

Attributes

Specifies servlet context attributes for the entire Web application.

- property Name - A servlet parameter of your choice
- property Value - The value associated with the property name

Note, the servlet context established by this property differs from the Shared Context,which pertains to clustering situations.

See the JSP administration overview for a descriptionof attributes related to JSP reloading, available starting with Version 3.5.2.

Auto Reload

Specify whether to automatically reload servlets in the Web application when their classfiles change.

After specifying to Auto Reload, use the Reload Interval property to specify how oftento check for updates.

Classpath

Specifies the classpath for the Web application.

Classpath in use

Specifies the classpath currently in use for the Web application.

Current State

Indicates the state the Web application is currently in. The next time itis started, it will try to change to its desired state setting.

Desired State

Indicates the state the Web application is in, according to the administrative server.

Description

Specifies a description of the Web application.

Document Root

Specifies the document root of the Web application.

Enabled

Indicates whether the servlet group (Web application) is available to handle requests.

Error Page FP 3.5.2 (changed)

Specifies mappings between error codes or exception types and the pathsof resources in the Web application. Basically, defines what to display tothe user in the event of a specific error.

Consists of:

- Status Code or Exception - An HTTP error code (such as 404) or fully qualified classname of a Java exception type
- Location - Location (in the Web application) of the error page to display when that status code or exception occurs

Example values:

- Status Code: 404

- Exception: java.lang.NullPointerException
- Location: /webapp/myapp/my404ErrorPage.jsp

The location is a "Web path," to use the terminology of the WebSphere Administrative Console.

Full Web Path in use

Specifies the URI by which the Web application can currently be located.

MIME Table FP 3.5.2

Specifies mappings between extensions and MIME types. Consists of:

- Extension - Text string describing an extension, such as .txt
- Type - The defined MIME type associated with the extension, such as text/plain

You can also specify MIME table parameters at the virtual host level,but the MIME table parameters you specify for a Web application takeprecedence (local scope).

Reload Interval

Specifies the interval between reloads of the web application.

Specify the value in *seconds*

6.6.8.1: Administering Web applications with the Java administrative console

This article extends article 6.6.8 (the overview of administering Web applications) with information specific to the Java console.

The table answers the most basic questions. See the Related informationfor links to detailed instructions and resource properties.

Does the console provide full functionality for administering this resource?	Yes
How is this resource representedin the console tree views?	The Type tree contains a Web Applications folderobject. The Topology tree can contain zero or moreexisting Web applications. Their names vary;they are supplied by the administrator. Use the View menu on the console menu bar to toggle between tree views.
Any task wizards for manipulatingthis resource?	On the console menu bar: Console -> Task -> Configure a Web application There are also subtasks: ● Add a servlet ● Add a JSP file or Web resource ● Add a JSP enabler

6.6.8.1.1: Configuring new Web applications

The product offers several ways to configure new Web applications:

- By clicking Console -> Tasks -> Configure a Web application from theconsole menu bar.
- By clicking Configure a Web application from the drop-down list on theWizards toolbar button.
- Using menus on resources in the Topology and Type trees(see Related information)

The first two methods lead to the Configure a Web applicationtask wizard, for which detailed help is provided here.

1. Follow the wizard instructions. On the first page, name the Webapplication and specify whetherto add WebSphere "internal" servlets to the Web applicationto perform certain functions:
   - File servlet
   - Enable Serving Servlets by Classname (adds invoker servlet)
   - JSP enabler (adds the JSP processor servlets)
   - Chainer servlet

   See article 4.2.1.2.3 for a detailed description ofeach internal servlet.
2. Click Next to proceed. Specify the servlet engine on which theWeb application should reside.
3. Click Next to proceed. Now:
   - Specify a name by which to administer the Web application.
   - Optionally, describe the Web application.
   - Specify the virtual host part of the Web application's served path. That is, what host name (or its aliases) will users specify when they access the Web application from a Web browser?
   - Specify the Web Path for the Web application. That is, what should users type in after the host name when requesting this Web application?

     For example, if you would like users to type

     `http://default_host_alias/webapp/mywebapp`

     to access the application (where *default_host_alias* is any valid alias for the default virtual host), specify:
     - Virtual Host = default_host_alias
     - Web Application Web Path= /webapp/mywebapp
4. Click Next to proceed:
   - Specify the document root for the Web application. This is the fully qualified path to where the HTML and JSP files for the Web application will be found.
   - Specify the classpath, adding either a directory for servlets or specifying servlets individually. Also specify any other resources the Web application needs to know about in order to operate correctly.

     Note that both the document root and the classpath contain default values. You can accept the default values and then move your files there after finishing the task. Alternatively, you can change the default values to point to your files in their present locations, or a location to which you plan to move them.
   - Specify other Web application properties or accept the default values for them.
5. Click Finish.

6.6.8.1.6: Converting WAR files with the Java administrative console

To convert WAR files (see article 0.8.2 for a description) using the Javaconsole:

1. Select the Convert WAR File task from the console Tasks menu.
2. Follow the instructions in the task wizard.

You will need to specify the following information:

- The servlet engine where the Web application will reside
- A name for the Web application
- A Web Path for the Web application
- The path to the WAR file

6.6.8.3: Administering Web applications with the Web console

Use the Web console to edit the configurations of Web applications.

Work with objects of this type by locating them in the tree on the left side of the console:

Click **Tasks** -> **Create Objects** -> **Create Web Application**

When creating a Web application, you must specifyan existing servlet engine to contain it. Existing Web applications andapplication servers in the administrative domain are displayed in the Resources section of the navigation tree.

 Creations and changes made with this console are not appliedto the administrative domain until you Commit them. Refer to section 6.6.0.3.5 for details.