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`

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:
   6.6.1.5.1: Creating an application

You can also compile your servlet using the `-classpath` option on the `javac` compiler. To access the classes that were extended, reference the`j2ee.jar` file in the *`product_installation_root`*`\lib`directory. Using this method, you issue the following command to compile your servlet:

```
javac -classpath product_installation_root\lib\j2ee.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:

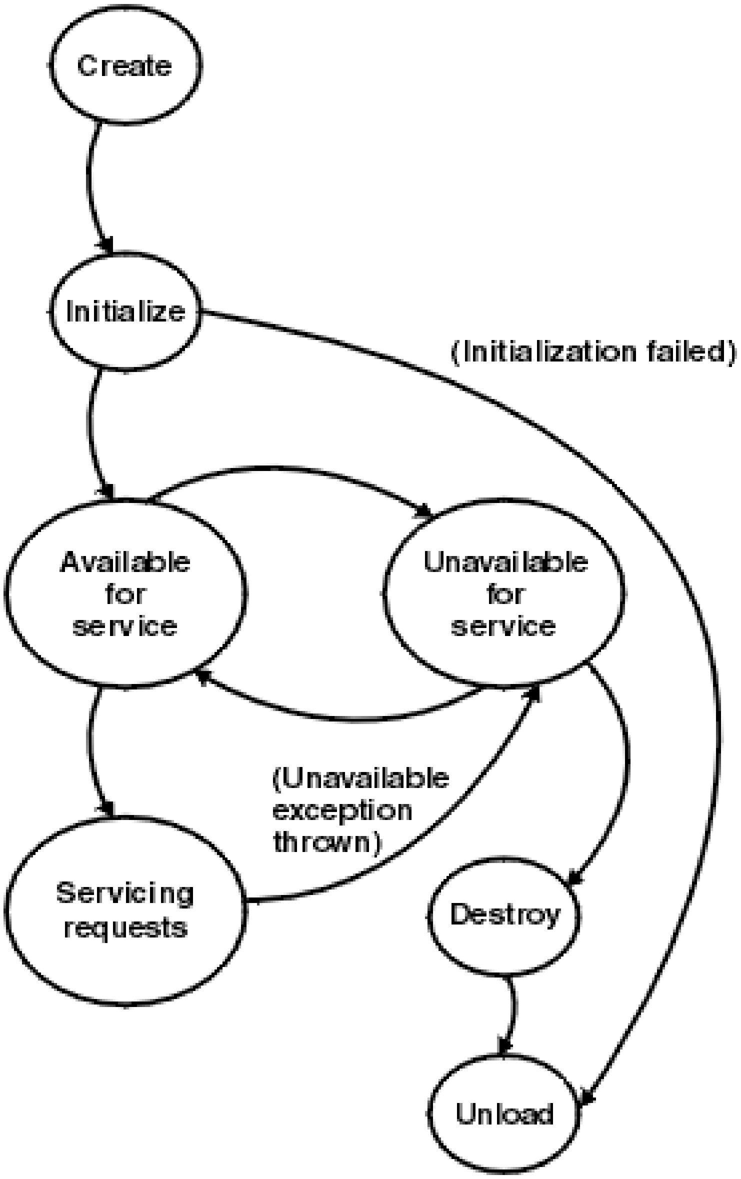
- Initialize a servlet
- Service requests
- Remove a servlet from the server

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 Web container (the Application Server entity that processes servlets, JSP files, and other types of server-side include coding) creates an instance of the servlet. The Web container creates the servlet configuration object and uses it to pass the servlet initialization parameters to the init method. The servlet configuration object persists 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 Web container 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 Web container creates a request object and a response object. The Web container 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 Web container 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 Web container 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 Web container loads 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 Web container 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.

See article 4.2.1.2.1afor a description of the Servlet API 2.2 specification.

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

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.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.

4.2.1.2.3a: Invoking servlets by classname and serving files

IBM Application Server provides some optional functions for your Web applications.

The tables below describe the function and how to use the WebSphere ApplicationServer tools to enable the function in your Web application.

Invoke servlets by class name

Objective	Invoke servlets by class or code names (such as MyServletClass)
How to enable the function	Use one of the following facilities: - If using the Application Assembly Tool (AAT),click **serve servlets by classname** in the IBM Extensions panel. - In the `ibm-web-ext.xmi` file, change the **serveServletsByClassnameEnabled** flag from *false* to *true*. The `ibm-web-ext.xmi` file is in the **WEB-INF** directory of theWeb module.

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 explicitlyconfigure the servlets in the WebSphere administrative domain.
How to enable the function	Use one of the following facilities: - If using the Application Assembly Tool (AAT),click **File Serving Enabled** in the IBM Extensions panel. - In the `ibm-web-ext.xmi` file, change the **fileServingEnabled** flag from *false* to *true*.

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.

Protecting servlets

See the article, Securing Applications, for information on securing servlets and Web resources.

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.
- **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 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");
```

4.2.1.3.2.2: 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>
```

4.2.1.3.4: 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, you must use an IBM supplied servlet named`com.ibm.websphere.servlet.filter.ChainerServlet`

1. Add the `com.ibm.websphere.servlet.filter.ChainerServlet` to your Web application during the application assembly stage and assign a servlet URL to the servlet instance.
2. 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

3. To invoke a servlet chain, invoke the servlet URL of the ChainerServlet in your application.

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()}
```

4.2.1.3.6: Serving servlets by classname

To enable serving servlets by classname, you can either:

- Click **serve servlets by classname** in the IBM Extensions panel of the Application Assembly Tool (AAT), or
- Change the **serveServletsByClassnameEnabled** flag in the `ibm-web-ext.xmi` file from*false* to *true*.

> The `ibm-web-ext.xmi` file is located in the **WEB-INF** directory of the installed Web module

See section 4.2.1.2.3a for details and instructions.

4.2.1.3.7: Serving all files from application servers

Files are served on a *per-web* module, not a *per-appserver* basis.To enable file serving, you can either:

- Click the **File Serving Enabled** checkbox in the IBM extensions panel of the Application Assembly Tool (AAT), or
- Change the **fileServingEnabled** flag from*false* to *true* in the `ibm-web-ext.xmi` file.

> ℹ The `ibm-web-ext.xmi` file is located in the **WEB-INF** directory of the installed Web module

See section 4.2.1.2.3a for 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.

4.2.1.3.9: PageListServlet support

IBM WebSphere Application Server supplies the PageListServlet to call a Java Server Page (JSP) by name. The PageListServlet uses configuration information to map a JSP name to the URI, where the URI specifies a JSP file in the WAR module. This support allows application developers to stop hard-coding URLs in their servlets.

These mappings, or page lists, are logically grouped according to the markup-language type (HTML, WML, and others) the JSP file is going to return to the requesting client. This allows applications, through the use of servlets that extend the PageListServlet class, to call a JSP file that returns the proper markup-language data type of the calling client. For example, if a request comes from a PDA, which requires WML data, and makes a request to a servlet that extends the PageListServlet, then a Java Server Page that returns WML data is called.

PageListServlet configuration information can be defined either in the IBM Web Extensions file or in an XML servlet configuration file. The IBM Web Extensions file is created and stored in the WAR file by the IBM WebSphere Application Assembly Tool. An XML servlet configuration file can be created using IBM WebSphere Studio or manually.

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

The `callPage()` method can be invoked as follows:

1. `callPage(String pageName, HttpServletRequest request, HttpServletResponse response)`
   - **pageName** - A page name defined in the PageListServlet configuration
   - **request** - The HttpServletRequest object
   - **response** - The HttpServletResponse object

   ℹ For this method of invocation, the default markup-language is HTML.
2. `callPage(String mlName, String pageName, HttpServletRequest req, HttpServletResponse resp)`
   - **mlName** - A markup-language type.
   - **pageName** - A page name defined in the PageListServlet configuration
   - **request** - The HttpServletRequest object
   - **response** - The HttpServletResponse object

See the Javadoc for the PageListServlet for a complete list of available APIs.

In addition to providing the page list mapping capability, the PageListServlet also has **Client Type Detection** support. Using the configuration information in the `client_types.xml` file, a servlet can determine the markup-language type the calling client requires for the response. This support allows the user's servlet to call an appropriate JSP, based on markup-language type. Use the second version of the `callPage()` method (described above) for **Client Type Detection** support.

In structuring the servlet code, keep in mind that the `PageListServlet.callPage()` method is not an exit. Any servlet code that follows this method call will be executed.

4.2.1.3.9.1: Extending PageListServlet

The `HelloPervasiveServlet` is an example of a servlet that extends the `PageListServlet` class and attempts to determine the markup-language typerequired by the client. The servlet then uses the `callPage()` method to callthe JSP with the page name of "Hello.page".

```
public class HelloPervasiveServlet extends PageListServlet implements Serializable{  /*   * doGet --
Process incoming HTTP GET requests   */      public void doGet(HttpServletRequest request,
HttpServletResponse response)  throws IOException, ServletException  {    // This is the name of the
page to be called.    String pageName = "Hello.page";        //  First check if the servlet was
invoked with a queryString     //  that  contained a markup-language value. For example, if this
//   servlet was  invoked like this:   //   http://localhost/servlets/HelloPervasive?mlname=VXML
String mlName = getMLNameFromRequest(request);    // If no ML type was provided in the queryString,
then attempt to    // determine the client type from the Request and use the ML name as    //
configured in the client_types.xml file.    if (mlName == null)    {      mlName =
getMLTypeFromRequest(request);    }    try    {      // Serve the Request page.
callPage(mlName, pageName, request, response);    }    catch (Exception e)    {
handleError(mlName, request, response, e);    }  }}
```

4.2.1.3.9.2: Configuring page lists using the Application Assembly Tool

PageListServlet configuration information can be defined in the IBM Web Extensions file or in an XML servlet configuration file. The IBM Web Extensions file is created and stored in the WAR file by the IBM WebSphere Application Assembly Tool (AAT). In the AAT, the page list information is configured under **PageList Extensions**.

4.2.1.3.9.3: Configuring page lists using an XML servlet configuration file

An alternative or legacy way of providing PageListServlet configuration information, is using an XML file known as the **XML Servlet Configuration** file. This file provides configuration information for page lists, and additional servlet configuration information. The file has a **`.servlet`** extension and resides in the same directory as the servlet class file. The XML servlet configuration file must be created with one of the following names:

1. `servlet_class_name.servlet`
2. `servlet_name.servlet`

IBM WebSphere Studio provides wizards that generate servlets with accompanying XML servlet configuration files.If you are not using IBM WebSphere Studio, you can manually create XML servlet configuration files.Each XML configuration file must be a well-formed XML document. The files are not validated against a Document Type Definition (DTD). Althoughthere is no DTD, it is recommended that all elements in the file appear in the same order as the elements described below:

XML Servlet configuration file elements

Elements	Description
servlet	The root element of an XML servlet configuration file.
code	The class name of the servlet, that extends the PageListServlet, without the `.class` extension.
description	The description of the servlet.
init-parameter	The attributes of this element specify the name-value pair to be used as an initialization parameter on the servlet. A servlet can have multiple initialization parameters, each within its own init-parameter element.
markup-language	Contains <ml-name>, <ml-mime>, and <page-list> elements. (The root element <servlet> can contain multiple <markup-language> elements.)
ml-name	A markup-language type, as for example: HTML, or WML, or VXML, and so forth
ml-mime	A MIME type, as for example: `text.html`, or `text/x-vxml`, or `text/vnd.wap.wml`, and so forth
page-list	Contains <default-page>, <error-page>, and <page>+ elements. (A <page-list> element can contain multiple <page> elements.)
default-page	Contains a <uri> element. The URI specifies the JSP to be called if the requested page does not exist or is not specified on the HTTP request.
error-page	Contains a <uri> element. The URI specifies the JSP to be called when the `handleError()` PageListServlet method is called.
page	Contains a <uri> and <page-name> element. The URI specifies the JSP file to be called when a PageListServlet method `callPage()` is called with the same value as <page-name>.
uri	A JSP file within the WAR Module.
page-name	The name in which a servlet, extending the PageListServlet, will use in the `callPage()` method to call a JSP.

4.2.1.3.9.4: Example of the XML servlet configuration file

```
<?xml version="1.0"?><servlet>  <code>HelloPervasiveServlet</code>  <description>Shows how to use
PageListServlet class.<:/description>  <init-parameter name="name1" value="value2"/>
<markup-language>    <ml-name>HTML</ml-name>    <ml-mime>text/html</ml-mime>    <page-list>
<error-page>        <uri>/mywebapp/HelloHTMLError.jsp</uri>      </error-page>      <page>
<page-name>Hello.page</page-name>        <uri>/mywebapp/HelloHTML.jsp</uri>      </page>
</page-list>  </markup-language>  <markup-language>    <ml-name>VXML</ml-name>
<ml-mime>text/x-vxml</ml-mime>    <page-list>      <error-page>
<uri>/mywebapp/HelloVXMLError.jsp</uri>      </error-page>      <page>
<page-name>Hello.page</page-name>        <uri>/mywebapp/HelloVXML.jsp</uri>      </page>
</page-list>  </markup-language>  <markup-language>    <ml-name>WML</ml-name>
<ml-mime>text/vnd.wap.wml</ml-mime>    <page-list>      <error-page>
<uri>/mywebapp/HelloWMLError.jsp</uri>      </error-page>      <page>
<page-name>Hello.page</page-name>        <uri>/mywebapp/HelloWML.jsp</uri>      </page>
</page-list>  </markup-language></servlet>
```

4.2.1.3.9.5: PageListServlet client type configuration file

In addition to providing the page list mapping capability, the PageListServlet also has **Client Type Detection** support. Using the configuration information in the `client_types.xml` file, a servlet can determine the markup-language type the calling client requires for the response.

This support allows the servlet, extending PageListServet, to call an appropriate JSP file,with the `callPage()` method, based on the markup-language type of the request. The client type detection method, `getMLTypeFromRequest(HttpServletRequest request)`, provided by the PageListServlet, inspects the HttpServletRequest object's request headers, and searches for a match in the `client_types.xml` file.

The client type detection method does the following:

1. Using the input HttpServletRequest and the `client_types.xml` file, it checks for a matching HTTP request name and value. If found, it returns the markup-language value configured for the `<client-type>` element.
2. If multiple matches are found, it returns the markup-language for the first `<client-type>` (for which a match was found).
3. If no match was found, it returns the value of the markup-language for the default page defined in the PageListServlet configuration.

The `client_types.xml` file is located in the *`product_installation_root`*`/properties` directory.

4.2.1.3.9.6: Example of a client type configuration file

```
<?xml version="1.0"?><!DOCTYPE clients [<!ELEMENT client-type
(description,markup-language,request-header+)><!ELEMENT description (#PCDATA)><!ELEMENT
markup-language (#PCDATA)><!ELEMENT request-header (name,value)><!ELEMENT clients
(client-type+)><!ELEMENT name (#PCDATA)><!ELEMENT value (#PCDATA)>]><clients>  <client-type>
<description>IBM Speech Client</description>    <markup-language>VXML</markup-language>
<request-header>      <name>user-agent</name>      <value>IBM VoiceXML pre-release version
000303</value>    </request-header>    <request-header>      <name>accept</name>
<value>text/vxml</value>    </request-header>  </client-type>  <client-type>    <description>WML
Browser</description>    <markup-language>WML</markup-language>    <request-header>
<name>accept</name>      <value>text/x-wap.wml</value>    </request-header>        <request-header>
<name>accept</name>      <value>text/vnd.wap.wml</value>    </request-header>
</client-type></clients>
```

6.6.7: Administering Web containers (overview)

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

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

6.6.7.0: Web container properties

Web containers contain other resource types, whose properties are listedin separate property reference files. If you do not find a property in the followinglist, see below for links to the property references of other resource types comprising Web containers.

Key:

☕ Applies to Java administrative console of Advanced Edition Version 4.0

🌐 Applies to Web administrative console of Advanced Single Server Edition Version 4.0

C:\ Applies to Application Client Resource Configuration Tool

Allow thread allocation beyond maximum ☕

Allows the number of threads to increase beyond the maximum size configured for the thread pool

Application Server ☕

The application server associated with this Web container

Cache Size ☕ 🌐

A positive integer defining the maximum number of entries the cache will hold. Values are usually in the thousands, with no maximum or minimum.

Can Be Grown 🌐

Allows the number of threads to increase beyond the maximum size configured for the pool

Default Priority ☕ 🌐

The default priority for servlets that can be cached. It determines how long an entry will stay in a full cache. The recommended value is 1.

Dynamic Properties 🌐

A set of name-value pairs for configuring properties beyond those displayed in the interface

Enable Dynamic Cache 🌐 **or Enable Servlet Caching** ☕

Enable the servlet and JSP dynamic JNDI caching feature

External Cache Groups ☕

For servlets that can be cached, specifies the external groups to which their entries should be sent

Enable Servlet Caching ☕

See Enable Dynamic Cache

HTTP Transports 🌐 **or Transport** ☕

The HTTP transports associated with this Web container. See also transport properties

Inactivity Timeout 🌐 **or Thread Inactivity Timeout** ☕

The period of time after which a thread should be reclaimed due to inactivity

Installed Web Modules

The Web modules that are installed into the Web container of this server

Maximum Size **or Maximum Thread Size**

The maximum number of threads to allow in the pool

Minimum Size **or Minimum Thread Size**

The minimum number of threads to allow in the pool

Name (External Cache Group)

See External Cache Groups

Node

The node with which this Web container is associated

Session Manager

The Session Manager associated with this Web container. See also Session Manager properties

Thread Inactivity Timeout

See Inactivity Timeout

Thread Pool

The thread pool settings for the Web container

Transport

See HTTP Transports

Type

Only shared external cache groups are supported at this time

Additional properties related to Web containers

Web containers contain other resource types, whose properties are listedin separate property reference files. If you do not find the property in theabove list ...

See also the:

- application server properties
- HTTP Transport properties

For Advanced Single Server Edition, see also the:

- Session Manager properties
- Web module properties

6.6.7.1: Administering Web containers with the Java administrative console

Use the Java administrative console to administer Web containers.

Work with resources of this type by locating them in the application server properties:

1. Locate the application server instance containing the service.
2. When the application server properties are displayed in the properties view, click the **Services** tabbed page.
3. Locate the Web container service in the list of services.

6.6.7.1.1: Configuring the Web container services of application servers with the Java administrative console

During this task, you will specify settings for the Web container service ofan existing application server.

1. Locate the Web container service in the application server properties.
2. Click **Edit Properties** to display the container service properties dialog.
3. Specify values for the Web container service properties.
4. When finished, click **OK**.

6.6.7.1.2: Administering transports of Web container services of application servers with the Java administrative console

HTTP transports that the Web server uses to relay requests to the applicationserver are accessed through the Web container configuration. See administering transports with the Java administrative consolefor a full description of settings and tasks.

6.6.8: Administering Web modules (overview)

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
```

To identify a different welcome page, modify the properties of theWeb module when you are assembling it. See the article aboutassembling Web modules with the ApplicationAssembly Tool (article 6.6.8.5).

Web application URLs are now case-sensitive on all operating systems

Please note that in Version 4.0.x, Webapplication URLs are now case-sensitive on all operating systems,for security and consistency.

For example, suppose you have a Web client application that runs successfully on Version 3.5.x. When running the same application on Version 4.0, you encounter an error that the welcome page (typically index.html), or HTML files to which it refers, cannot be found:

```
    Error 404: File not found: Banner.html     Error 404: File not found: HomeContent.html
```

Suppose the content of the index page is as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN"><HTML><TITLE>Insurance Home Page</TITLE>
<frameset   rows="18,80">    <frame      src="Banner.html"          name="BannerFrame"
SCROLLING=NO>     <frame      src="HomeContent.html"      name="HomeContentFrame">
</frameset></HTML>
```

but the actual file names in \WebSphere\AppServer\installedApps\... directory in whichthe application is deployed are:

```
banner.htmlhomecontent.html
```

To correct the problem, modify the index.html file to change the names "Banner.html"and "HomeContent.html" to "banner.html" and "homecontent.html" to match the names of thefiles in the deployed application.

6.6.8.0: Web module properties

Key:

Applies to Java administrative console of Advanced Edition Version 4.0

Applies to Web administrative console of Advanced Single Server Edition Version 4.0

Applies to Application Client Resource Configuration Tool

Application or Application Ref

The application installation binding within which the module-to-server installation binding is contained. This is typically the logical name of the enterprise application you configured to contain this Web module.

Context Root

The context root of the Web application contained in an enterprise application.

The context root is combined with the defined servlet mapping (from the WAR file) to compose the full URL that users type to access the servlet. For example, if the context root is /gettingstarted and the servlet mapping is MySession, then the URL is http://host:port/gettingstarted/MySession.

Execution State

The state that you would like the Web module to be in, the next time the product is stopped and started again

Name or Module Name

An administrative name for the Web module

Server

The application server on which the Web module is installed

URI

A URI that, when resolved relative to the application URL, specifies the location of the module archive on a file system. The URI must match the URI of a ModuleRef URI in the deployment descriptor of an application if the module was packaged as part of a deployed application (EAR).

6.6.8.0.1: Assembly properties for Web components

Component name (Required, String)

Specifies the name of the servlet or JavaServer Pages(TM) (JSP)file. This name must be unique within the Web module.

Display name

Specifies a short name that is intended to be displayed by GUIs.

Description

Contains a description of the servlet or JSP file.

Component type

Specifies the type of Web component. Valid values are servlet orJSP file.

Class name (Required, String)

Specifies the full path name for the servlet's class.

JSP file (Required, String)

Specifies the full path name for the JSP file.

Load on startup

Indicates whether this servlet is to be loaded at the startup of the Webapplication. The default is false (the checkbox is notselected). Also specifies a positive integer indicating the order inwhich the servlet is to be loaded. Lower integers are loaded beforehigher integers. If no value is specified, or if the value specified isnot a positive integer, the container is free to load the servlet at any timein the startup sequence.

Small icon

Specifies a JPEG or GIF file containing a small image (16x16pixels). The image is used as an icon to represent the Web component ina GUI.

Large icon

Specifies a JPEG or GIF file containing a large image (32x32pixels). The image is used as an icon to represent the Web component ina GUI.

6.6.8.0.2: Assembly properties for initialization parameters

Initialization parameters are sent to a servlet in its HttpConfig objectwhen the servlet is first started.

Parameter name (Required, String)

Specifies the name of an initialization parameter.

Parameter value (Required, String)

Specifies the value of the initialization parameter.

Description

Contains text describing the use of the parameter.

6.6.8.0.3: Assembly properties for page lists

Page lists allow you to avoid hardcoding URLs in servlets and JSPfiles. A page list specifies the location where a request is to beforwarded, but automatically tailors that location depending on the MIME typeof the servlet. These properties allow you to specify a markup languageand an associated MIME type. For the given MIME type, you also specifya set of pages to invoke. For example, if you define a markup languagenamed VXML and associate it with a vxml MIME type, you can then define Pagenames and URIs to be invoked for that particular MIME type. The Pagenames end in .page and are the same name for all markuplanguages. However, the URIs are set to point to files that areparticular to the given MIME type. For example, if a page is calledShowAccount.page and is in a markup language named VXML, the URI isShowAccountVXML.jsp. In a markup language named HTML, the URI isShowAccountHTML.jsp. When the servlet refers toShowAccount.page, the actual file to which the request maps depends onthe servlet's MIME type.

Name

Specifies the name of the markup language--for example, HTML, WML,and VXML.

MIME Type

Specifies the MIME type of the markup language--for example,text/html and text/x-vxml.

Error Page

Specifies the name of an error page.

Default Page

Specifies the name of a default page.

Pages - Name

Specifies the name of the page to be served, for example,StockQuoteRequest.page.

Pages - URI

Specifies the URI of the page to be served, for example,examples/StockQuoteHTMLRequest.jsp.

6.6.8.0.4: Assembly properties for security constraints

Security constraints declare how Web content is to be protected.These properties associate security constraints with one or more Web resourcecollections. A constraint consists of a Web resource collection, anauthorization constraint, and a user data constraint.

- A Web resource collection is a set of resources (URL patterns) and HTTPmethods on those resources. All requests that contain a request paththat matches the URL pattern described in the Web resource collection issubject to the constraint. If no HTTP methods are specified, then thesecurity constraint applies to all HTTP methods.
- An authorization constraint is a set of roles that users must be grantedin order to access the resources described by the Web resourcecollection. If a user who requests access to a specified URI is notgranted at least one of the roles specified in the authorization constraint,the user is denied access to that resource.
- A user data constraint indicates that the transport layer of theclient/server communications process must satisfy the requirement of eitherguaranteeing content integrity (preventing tampering in transit) orguaranteeing confidentiality (preventing reading while in transit).

If multiple security constraints are specified, the container uses the"first match wins" rule when processing a request to determine whatauthentication method to use, or what authorization to allow.

Security constraint name

Specifies the name of the security constraint.

Authorization Constraints - Roles

Specifies the user roles that are permitted access to this resourcecollection.

Authorization Constraints - Description

Contains a description of the authorization constraints.

User Data Constraints - Transport guarantee

Indicates how data communicated between the client and the server is to beprotected. Specifies that the protection for communications between theclient and server is None, Integral, or Confidential. None means thatthe application does not require any transport guarantees. Integralmeans that the application requires that the data sent between the client andthe server must be sent in such a way that it cannot be changed intransit. Confidential means that the application requires that the datamust be transmitted in a way that prevents other entities from observing thecontents of the transmission. In most cases, Integral or Confidentialindicates that the use of SSL is required.

User Data Constraints - Description

Contains a description of the user data constraints.

6.6.8.0.5: Assembly properties for Web resource collections

A Web resource collection defines a set of URL patterns (resources) andHTTP methods belonging to the resource. HTTP methods handle HTTP-basedrequests, such as GET, POST, PUT, and DELETE. A URL pattern is apartial Uniform Resource Locator that acts as a template for matching thepattern with existing full URLs in an attempt to find a valid file.

Web resource name (Required, String)

Specifies the name of a Web resource collection.

Web resource description

Contains a description of the Web resource collection.

HTTP methods

Specifies the HTTP methods to which the security constraintsapplies. If no HTTP methods are specified, then the security constraintapplies to all HTTP methods. The valid values are GET, POST, PUT,DELETE, HEAD, OPTIONS, and TRACE.

URL pattern

Specifies URL patterns for resources in a Web application. Allrequests that contain a request path that matches the URL pattern are subjectto the security constraint.

6.6.8.0.8: Assembly properties for context parameters

A servlet context defines a server's view of the Web applicationwithin which the servlet is running. The context also allows a servletto access resources available to it. Using the context, a servlet canlog events, obtain URL references to resources, and set and store attributesthat other servlets in the context can use. These properties declare aWeb application's parameters for its context. They convey setupinformation, such as a webmaster's e-mail address or the name of a systemthat holds critical data.

Parameter name (Required, String)

Specifies the name of a parameter--for example,dataSourceName.

Parameter value (Required, String)

Specifies the value of a parameter--for example, jdbc/sample.

Description

Contains a description of the context parameter.

6.6.8.0.9: Assembly properties for error pages

Error page locations allow a servlet to find and serve a URI to a clientbased on a specified error status code or exception type. Theseproperties are used if the error handler is another servlet or JSPfile. The properties specify a mapping between an error code orexception type and the path of a resource in the Web application. Thecontainer examines the list in the order that it is defined, and attempts tomatch the error condition by status code or by exception class. On thefirst successful match of the error condition, the container serves back theresource defined in the Location property.

Error Code

Indicates that the error condition is a status code.

Error Code (Required, String)

Specifies an HTTP error code, for example, 404.

Exception

Indicates that the error condition is an exception type.

Exception type name (Required, String)

Specifies an exception type.

Location (Required, String)

Contains the location of the error-handling resource in the Webapplication.

6.6.8.0.10: Assembly properties for MIME mapping

A Multi-Purpose Internet Mail Extensions (MIME) mapping associates a filename extension with a type of data file (text, audio, image). Theseproperties allow you to map a MIME type to a file name extension.

Extension (Required, String)

Specifies a file name extension, for example, .txt.

MIME type (Required, String)

Specifies a defined MIME type, for example, text/plain.

6.6.8.0.11: Assembly properties for servlet mapping

A servlet mapping is a correspondence between a client request and aservlet. Servlet containers use URL paths to map client requests toservlets, and follow the URL path-mapping rules as specified in the JavaServlet specification. The container uses the URI from the request,minus the context path, as the path to map to a servlet. The containerchooses the longest matching available context path from the list of Webapplications that it hosts.

URL pattern (Required, String)

Specifies the URL pattern of the mapping. The URL pattern mustconform to the Servlet specification. The following syntax must beused:

- A string beginning with a slash character (/) and ending with the slashand asterisk characters (/*) is used as a path mapping.
- A string beginning with the characters *. is used as an extensionmapping.
- All other strings are used as exact matches only.
- A string containing only the slash character (/) indicates that theservlet specified by the mapping becomes the default servlet of theapplication. In this case, the servlet path is the request URI minusthe context path, and the path info is null.

Servlet (Required, String)

Specifies the name of the servlet associated with the URL pattern.

6.6.8.0.12: Assembly properties for tag libraries

Java ServerPages (JSP) tag libraries contain classes for common tasks suchas processing forms and accessing databases from JSP files.

Tag library file name (Required, String)

Specifies a file name relative to the location of the web.xmldocument, identifying a tag library used in the Web application.

Tag library location (Required, String)

Contains the location, as a resource relative to the root of the Webapplication, where the Tag Library Definition file for the tag library can befound.

6.6.8.0.13: Assembly properties for welcome files

A Welcome file is an entry-point file (for example, index.html) fora group of related HTML files. Welcome files are located by using agroup of suggested partial URIs. A Welcome file is an ordered list ofpartial URIs that the container uses to attempt to find a valid file when theinitial URI cannot be found. The container appends these partial URIsto the requested URI to arrive at a valid URI. For example, the usercan define a Welcome file of index.html so that a request to a URL suchas host:port/webapp/directory (where directory is a directoryentry in the WAR file that is not mapped to a servlet or JSP file) can befulfilled.

Welcome file (Required, String)

The Welcome file list is an ordered list of partial URLs with no trailingor leading slash characters (/). The Web server appends each file inthe order specified and checks whether a resource in the WAR file is mapped tothat request URI. The container forwards the request to the firstresource in the WAR that matches.

6.6.8.0.14: Assembly properties for MIME filters

Filters transform either the content of an HTTP request or response and canalso modify header information. MIME filters forward HTTP responseswith a specified MIME type to one or more designated servlets for furtherprocessing.

MIME Filter - Target

Specifies the target virtual host for the servlets.

MIME Filter - Type

Specifies the MIME type of the response that is to be forwarded.

6.6.8.0.15: Assembly properties for JSP attributes

JSP attributes are used by the servlet that implements JSP processingbehavior.

JSP Attribute (Name)

Specifies the name of an attribute.

JSP Attribute (Value)

Specifies the value of an attribute.

6.6.8.0.16: Assembly properties for file-serving attributes

File serving allows a Web application to serve static file types, such asHTML. File-serving attributes are used by the servlet that implementsfile-serving behavior.

File Serving Attribute (Name)

Specifies the name of an attribute.

File Serving Attribute (Value)

Specifies the value of an attribute.

6.6.8.0.17: Assembly properties for invoker attributes

Invoker attributes are used by the servlet that implements the invocationbehavior.

Invoker Attribute (Name)

Specifies the name of an attribute.

Invoker Attribute (Value)

Specifies the value of an attribute.

6.6.8.0.18: Assembly properties for servlet caching configurations

Dynamic caching can be used to improve the performance of servlet andJavaServer Pages (JSP) files by serving requests from an in-memorycache. Cache entries contain the servlet's output, results of theservlet's execution, and metadata.

The properties on the General tab define a cache group and govern how longan entry remains in the cache. The properties on the ID Generation tabdefine how cache IDs are built and the criteria used to cache or invalidateentries. The properties on the Advanced tab define external cachegroups and specify custom interfaces for handling servlet caching.

Caching group name (Required, String)

Specifies a name for the group of servlets or JSP files to becached.

Priority

An integer that defines the default priority for servlets that arecached. The default value is 1. Priority is an extension of theLeast Recently Used (LRU) caching algorithm. It represents the numberof cycles through the LRU algorithm that an entry is guaranteed to stay in thecache. The priority represents the length of time that an entry remainsin the cache before being eligible for removal. On each cycle of thealgorithm, the priority of an entry is decremented. When the priorityreaches zero, the entry is eligible for invalidation. If an entry isrequested while in the cache, its priority is reset to the priorityvalue. Regardless of the priority value and the number of requests, anentry is invalidated when its timeout occurs. Consider increasing thepriority of a servlet or JSP file when it is difficult to calculate the outputof the servlet or JSP file or when the servlet or JSP file is executed moreoften than average. Priority values should be low. Higher valuesdo not yield much improvement but use extra LRU cycles. Use timeout toguarantee the validity of an entry. Use priority to rank the relativeimportance of one entry to other entries. Giving all entries equalpriority results in a standard LRU cache that increases performancesignificantly.

Timeout

Specifies the length of time, in seconds, that a cache entry is to remainin the cache after it has been created. When this time elapses, theentry is removed from the cache. If the timeout is zero or a negativenumber, the entry does not time out. It is removed when the cache isfull or programmatically, from within an application.

Invalidate only

Specifies that invalidations for a servlet are to take place, but that nocaching is to be performed for the servlet. For example, this propertycan be used to prevent caching of control servlets. Control servletstreat HTTP requests as commands and execute those commands. By default,this checkbox is not selected.

Caching group members

Specifies the names of the servlets or JSP files to be cached. TheURIs are determined from the servlet mappings.

Use URIs for cache ID building

Specifies whether or not the URI of the requested servlet is to be used tocreate a cache ID. By default, URIs are used.

Use specified string

Specifies a string representing a combination of request and sessionvariables that are to be used to create cache IDs. (This propertydefines request and session variables, and the cache uses the values of thesevariables to create IDs for the entries.)

Variables - ID

The name of a request parameter, request attribute, session parameter, orcookie.

Variables - Type

Indicates the type of variable specified in the ID field. The validvalues are Request parameter, Request attribute, Session parameter, orCookie.

Variables - Method

The name of a method in the request attribute or session parameter.The output of this method is used to generate cache entry IDs. If thisvalue is not specified, the toString method is used by default.

Variables - Data ID

Specifies a string that, combined with the value of the variable,generates a group name for the cache entry. The cache entry is placedin this group. This group can later be invalidated.

Variables - Invalidate ID

Specifies a string that is combined with the value of the variable on therequest or session to form a group name. The cache invalidates thegroup name.

Required

Indicates whether a value must be present in the request. If thischeckbox is selected, and either the request parameter, request attribute, orsession parameter is not specified, or the method is not specified, therequest is not cached.

External cache groups - Group name

Specifies the name of the external cache group to which this servlet willbe published.

ID generator

Specifies a user-written interface for handling parameters, attributes,and sessions. The value must be a full package and class name of aclass extendingcom.ibm.websphere.servlet.cache.IdGenerator.The properties specified in the Application Assembly Tool will still be usedand passed to the IdGenerator in the initialize method inside acom.ibm.websphere.servlet.cache.CacheConfigobject.

Meta data generator

Specifies a user-written interface for handling invalidation, prioritylevels, and external cache groups. The value must be the full packageand class name of a class extendingcom.ibm.websphere.servlet.cache.MetaDataGenerator.The properties specified in the Application Assembly Tool will still be usedand passed to the MetaDataGenerator in the initialize method inside acom.ibm.websphere.servlet.cache.CacheConfigobject.

6.6.8.0.aa: Assembly properties for Web modules

File name (Required, String)

Specifies the file name of the Web module, relative to the top level ofthe application package.

Alternative DD

Specifies the file name for an alternative deployment descriptor file touse instead of the original deployment descriptor file in the module'sJAR file. This file is the postassembly version of the deploymentdescriptor file. (The original deployment descriptor file can be editedto resolve dependencies and security information. Directing the use ofthe alternative deployment descriptor allows you to keep the originaldeployment descriptor file intact). The value of the Alternative DDproperty must be the full path name of the deployment descriptor file relativeto the module's root directory. By convention, the file is in theALT-INF directory. If this property is not specified, the deploymentdescriptor file is read directly from the module's JAR file.

Context root (Required, String)

Specifies the context root of the Web application. The context rootis combined with the defined servlet mapping (from the WAR file) to composethe full URL that users type to access the servlet. For example, if thecontext root is /gettingstarted and the servlet mapping is MySession, then theURL is http://host:port/gettingstarted/MySession.

Classpath

Specifies the full class path for the Web application. Specify thevalues relative to the root of the EAR file and separate the values withspaces. Absolute values that reference files or directories on the harddrive are ignored. To specify classes that are not in JAR files but arein the root of the EAR file, use a period and forward slash(./). Consider the following example directory structure inwhich the file myapp.ear contains a Web module namedmywebapp.war. Classes reside in class1.jar andclass2.zip. A class named xyz.class is not packaged in aJAR file but is in the root of the EAR file.

```
myapp.ear/mywebapp.warmyapp.ear/class1.jarmyapp.ear/class2.zipmyapp.ear/xyz.class
```

Specify `class1.jar class2.zip ./` as thevalue of the Classpath property. (Name only the directory for.class files.)

Display name

Specifies a short name that is intended to be displayed by GUIs.

Description

Contains a description of the Web module.

Distributable

Specifies that this Web application is programmed appropriately to bedeployed into a distributed servlet container.

Small icon

Specifies a JPEG or GIF file containing a small image (16x16pixels). The image is used as an icon to represent the module in aGUI.

Large icon

Specifies a JPEG or GIF file containing a large image (32x32pixels). The image is used as an icon to represent the module in aGUI.

Session configuration

Indicates that session configuration information is present.Checking this box makes the Session timeout property editable.

Session timeout

Specifies a time period, in seconds, after which a client is consideredinactive. The default value is zero, indicating that the sessiontimeout never expires.

Login configuration -- Authentication method

Specifies an authentication method to use. As a prerequisite togaining access to any Web resources protected by an authorization constraint,a user must authenticate by using the configured mechanism. A Webapplication can authenticate a user to a Web server by using one of thefollowing mechanisms: HTTP basic authentication, HTTP digestauthentication, HTTPS client authentication, and form-basedauthentication.

- HTTP basic authentication is not a secure protocol because the userpassword is transmitted with a simple Base64 encoding and the target server isnot authenticated. In basic authentication, the Web server requests aWeb client to authenticate the user and passes a string called the realm ofthe request in which the user is to be authenticated.

- HTTP digest authentication transmits the password in encryptedform.
- HTTPS client authentication uses HTTPS (HTTP over SSL) and requires theuser to possess a public key certificate.
- Form-based authentication allows the developer to control the appearanceof login screens.

The Login configuration properties are used to configure the authenticationmethod that should be used, the realm name that should be used for HTTP basicauthentication, and the attributes that are needed by the form-based loginmechanism. Valid values for this property are Unspecified, Basic,Digest, Form, and Client certification.

Note: HTTP digest authentication is not supported as a loginconfiguration in this product. Also, not all login configurations aresupported in all of the product's global security authenticationmechanisms (Local Operating system, LTPA, and custom pluggable userregistry). HTTP basic authentication and form-based loginauthentication are the only authentication methods supported by the LocalOperating system user registry. Because Advanced Single Server Editionuses the local operating system as the user registry for authentication, itcan only support these two login methods. LTPA and the custom pluggableuser registry are capable of supporting HTTP basic authentication, form-basedlogin, and HTTPS client authentication. LTPA and the custom pluggableuser registry is available only in Advanced Edition.

Login configuration -- Realm name

Specifies the realm name to use in HTTP basic authorization. It isbased on a user name and password, sent as a string (with a simple Base64encoding). An HTTP realm is a string that allows URIs to be groupedtogether. For example, if a user accesses a secured resource on a Webserver within the "finance realm," subsequent access to the same or differentresource within the same realm does not result in a repeat prompt for a userID and password.

Login configuration -- Login page (Required, String)

Specifies the location of the login form. If form-basedauthentication is not used, this property is disabled.

Form Login Config -- Error page (Required, String)

Specifies the location of the error page. If form-basedauthentication is not used, this property is disabled.

Reload interval

Specifies a time interval, in seconds, in which the Web application'sfile system is scanned for updated files. The default is 0(zero).

Reloading enabled

Specifies whether file reloading is enabled. The default isfalse.

Default error page

Specifies a file name for the default error page. If no other errorpage is specified in the application, this error page is used.

Additional classpath

Specifies an additional class path that will be used to referenceresources outside of those specified in the archive. Specify the valuesrelative to the root of the EAR file and separate the values withspaces. Absolute values that reference files or directories on the harddrive are ignored. To specify classes that are not in JAR files but arein the root of the EAR file, use a period and forward slash(./). Consider the following example directory structure inwhich the file myapp.ear contains a Web module namedmywebapp.war. Additional classes reside in class1.jar andclass2.zip. A class named xyz.class is not packaged in aJAR file but is in the root of the EAR file.

```
myapp.ear/mywebapp.warmyapp.ear/class1.jarmyapp.ear/class2.zipmyapp.ear/xyz.class
```

Specify `class1.jar class2.zip ./` as thevalue of the Additional classpath property. (Name only the directoryfor .class files.)

File serving enabled

Specifies whether file serving is enabled. File serving allows theapplication to serve static file types, such as HTML and GIF. Fileserving can be disabled if, for example, the application contains only dynamiccomponents. The default value is true.

Directory browsing enabled

Specifies whether directory browsing is enabled. Directory browsingallows the application to browse disk directories. Directory browsingcan be disabled if, for example, you want to protect data. The defaultvalue is true.

Serve servlets by classname

Specifies whether a servlet can be served by requesting its classname. Usually, servlets are served only through a URI reference.The class name is the actual name of the servlet on disk. For example,a file named SnoopServlet.java compiles intoSnoopServlet.class. (This is the class name.)SnoopServlet.class is normally invoked by specifying snoop in theURI. However, if Serve Servlets by Classname is enabled, the servlet isinvoked by specifying SnoopServlet. The default value is true.

Virtual hostname

Specifies a virtual host name. A virtual host is a configurationenabling a single host machine to resemble multiple host machines. Thisproperty allows you to bind the application to a virtual host in order toenable execution on that virtual host.

6.6.8.1: Administering Web modules with the Java administrative console

Work with resources of this type by locating them in the tree view.

To locate modules according to their membership in enterprise applications, click:

WebSphere Administrative Domain -> Nodes -> *node_name* -> Enterprise Applications -> *application_name* ->Web modules

To locate modules installed on a particular application server, click:

WebSphere Administrative Domain -> Nodes -> *node_name* -> Application Servers -> *application_server_name* ->Installed Web modules

In either case, the instances of the module will be displayed in thedetails view.

6.6.8.1.1: Installing Web modules with the Java administrative console

Use the application installation wizard to install Web modules onto an application server.

6.6.8.1.2: Viewing deployment descriptors of Web modules with the Java administrative console

To view the deployment descriptor of a Web module:

1. Locate the Web module instance.
2. Right-click your Web module in the details view and select **View Deployment Descriptor** from the popup menu.

A Web Module window opens, providing descriptive information about the selected module's Web.xml, servlets, servlet mappings, welcome file, security constraints and role, login configuration, and EJB reference.

6.6.8.1.3: Showing the status of Web modules with the Java administrative console

During this task, you will view the status (such as running or stopped) of each Web module in an enterprise application.

1. Locate the application instance.
2. Click the Installed EJB modules folder of the application to display its Web module instances in the details view.
3. Right-click an instance and select **Show Status** to display the Module Status window.
4. When finished viewing status, close the window.

6.6.8.1.5: Moving Web modules to other application servers with the Java administrative console

You can move a Web module from one application server to a different application server or server group. To move a module:

1. Locate the Web module instance.
2. Right-click your Web module in the details view and select **Move** from the popup menu.
3. In the Select a target server or ServerGroup dialog that opens, specify the target application server or server group and click **OK**
4. Copy WAR files containing the module to the node that contains the destination server. If the destination is a server group, then copy the WAR file to all nodes that contain the clones of the destination server group.

6.6.8.5: Administering Web modules with Application Assembly Tool

A Web module is used to assemble one or more servlets, JavaServer Pages(JSP) files, Web pages, and other static content into a single deployableunit. The Application Assembly Tool is used to create and edit modules,verify the archive files, and generate deployment code. See the relatedtopics for links to concepts, instructions for creating a Web module, andfield help.

6.6.8.5.1: Creating a Web module

Web modules can be created by using property dialog box or by using awizard.

- Using the property dialog boxes
- Using the Create Web Module wizard

Using the property dialog boxes

The steps for creating a Web module are as follows:

1. Click **File**->**New**->**Web Module**. Thenavigation pane displays a hierarchical structure used to build the contentsof the module. The icons represent the components, assembly properties,and files for the module. A property dialog box containing generalinformation about the module is displayed in the property pane.
2. By default, the archive file name and the module display name are thesame. It is recommended that you change the display name in theproperty pane. Enter values for other properties as needed. Viewthe help for 6.6.8.0.a: Assembly properties for Web modules.
3. By default, the temporary location of the Web module isinstallation_directory`/bin`. You must specify a newfile name and location by clicking **File**->**Save**.You must first add at least one Web component (servlet or JSP file) beforesaving the archive.
4. Add Web components (servlets or JSP files) to the module. You mustadd at least one Web component. There are several ways of addingcomponents to a module:
    - Import an existing WAR file containing Web components. In thenavigation pane, right-click the Web Components icon and choose**Import**. Click **Browse** to browse the file systemand locate the desired archive file. When the file is located, click**Open**. The Web applications in the selected archive file aredisplayed. Select a Web application. Its Web components aredisplayed in the right window. Select the servlets or JSP files to beadded and click **Add**. The components are displayed in theSelected Components window. Click **OK**. The propertiesassociated with the archive are also imported and the property dialog boxesare automatically populated with values. Double-click the WebComponents icon to verify that the servlets or JSP files are included in themodule.
    - Use a copy-and-paste operation to copy archive files from an existingmodule.
    - Create a new Web component. Right-click the Web Components icon andchoose **New**. Enter a component name and choose a componenttype. Browse for and select the class files. By default, theroot directory or archive is the current archive. If needed, browse thefile system for the directory or archive where the class files reside.After you choose a directory or archive, its file structure isdisplayed. Expand the structure and locate the files that youneed. Select the file and click **OK**. In the New WebComponent property dialog box, click **OK**. Verify that the Webcomponent has been added to the module (double-click the Web Components iconin the navigation pane). The Web components are also listed in the topportion of the property pane. Click the component to view itscorresponding property dialog box in the bottom portion of the pane.
5. Enter properties for the Web component as needed. View the help for6.6.8.0.1: Assembly properties for Web components.
6. Enter assembly properties for each Web component. Click the plussign (+) next to the component instance to reveal propertygroups. Right-click each property group's icon. Choose**New** to add new values, or edit existing values in the propertypane.
    - Specify Security Role References. View the help for 6.6.43.0.3: Assembly properties for security

role references.

  - Specify Initialization Parameters. View the help for 6.6.8.0.2: Assembly properties for initialization parameters.
  - Specify Page List Extensions. View the help for 6.6.8.0.3: Assembly properties for page lists.

7. Specify additional properties for the Web module. Right-click eachproperty group's icon. Choose **New** to add new values, oredit existing values in the property pane.
    - Specify Security Constraints. View the help for 6.6.8.0.4: Assembly properties for security constraints. If you add a security constraint, you must add atleast one Web resource collection.
    - Specify Web resource collections, HTTP methods, and URL patterns.View the help for 6.6.8.0.5: Assembly properties for Web resource collections.
    - Specify Context Parameters. View the help for 6.6.8.0.8: Assembly properties for context parameters.
    - Specify EJB references. View the help for 6.6.43.0.1: Assembly properties for EJB references.
    - Specify Environment Entries. View the help for 6.6.34.0.a: Assembly properties for environment entries.
    - Specify Error Pages. View the help for 6.6.8.0.9: Assembly properties for error pages.
    - Specify MIME Mappings. View the help for 6.6.8.0.10: Assembly properties for MIME mapping.
    - Specify Resource References. View the help for 6.6.43.0.2 Assembly properties for resource references.
    - Specify Security Roles. View the help for 6.6.5.0.5: Assembly properties for security roles.
    - Specify Servlet Mapping. View the help for 6.6.8.0.11: Assembly properties for servlet mapping.
    - Specify Tag Libraries. View the help for 6.6.8.0.12: Assembly properties for tag libraries.
    - Specify Welcome Files. View the help for 6.6.8.0.13: Assembly properties for welcome files.
8. Optionally, specify assembly property extensions. In the navigationpane, double-click the icon for Assembly Property Extensions.
    - Specify MIME filters. View the help for 6.6.8.0.14: Assembly properties for MIME filters.
    - Specify JSP Attributes. View the help for 6.6.8.0.15: Assembly properties for JSP attributes.
    - Specify File Serving Attributes. View the help for 6.6.8.0.16: Assembly properties for file-serving attributes.
    - Specify Invoker Attributes. View the help for 6.6.8.0.17: Assembly properties for invoker attributes.
    - Specify Servlet Caching Configurations. View the help for 6.6.8.0.18: Assembly properties for servlet caching configurations.
9. Add any other files needed by the application. In the navigationpane, click the plus sign (+) next to the Files icon.Right-click **Add Class Files**, **Add JAR Files**, or **AddResource Files**. Choose **Add Files**. Click**Browse** to navigate to the desired directory or archive and thenclick **Select**. If you are adding an entire archive, selectthe directory that contains the archive. The directory structure isdisplayed in the left pane. Browse the directory structure. Fromthe right pane, select one or more files to be added and click**Add**. If you select a directory and click **Add**, allfiles in the directory, including the directory, are added. Relativepath names are maintained. When the Selected Files window contains thecorrect set of files, click **OK**.

10. Click **File**->**Save** to save the archive.

Using the Create Web Module wizard

Use this wizard to create a Web module. The module can then be usedas a stand-alone application, or it can become part of a J2EE applicationcontaining other modules. A Web module consists of one or more servletsand JSP files. You can use existing archives (by importing them), orcreate new ones.

During creation of the Web module, you specify the files for each servletor JSP file to be included in the module. You also specify assemblyproperties for the servlets and JSP files, such as references to enterprisebeans and resource connection factories, and security roles. Thecontent information and assembly properties are used to create a deploymentdescriptor.

Before you start the wizard, you must have the required files for yourservlet or JSP file. When the wizard is completed, your Web module (WARfile) is created in the directory that you specify.

To create a Web module, click the **Wizards** icon on the tool barand then click **Web Module**. Follow the instructions on eachpanel.

- Specifying Web module properties
- Adding files
- Specifying optional Web module properties
- Choosing Web Module icons
- Adding Web components
- Adding security roles
- Adding servlet mappings
- Adding resource references
- Adding context parameters
- Adding error pages
- Adding MIME mappings
- Adding Tag Libraries
- Adding Welcome Files
- Adding EJB references
- Setting additional properties and saving the archive

Specifying Web module properties

On the **Specifying Web Module Properties** panel:

1. Indicate the application to which this module is to be added. If aparent application is not indicated, the module is created as a stand-aloneapplication.
2. Specify a file name and display name for the module. The displayname is used to identify your module in the Application Assembly Tool and canbe used by other tools. The file name specifies a location on yoursystem where the WAR file is to be created.
3. Provide a short description of the module.
4. Click **Next**.

Adding files

On the **Adding Files** panel, specify the files that are to beassembled for your Web module.

1. Click **Add Resource Files**, **Add Class Files**, or**Add JAR files**, depending on the type of file you are adding.First, browse for the root directory or archive where the files are locatedand click **Select**. If you are adding an entire archive,select the directory that contains the archive. The directory structureis displayed in the left pane. Browse the directory structure.From the right pane, select one or more files to be added and click**Add**. If you select a directory and click **Add**, allfiles in the directory, including the directory, are added. Relativepath names are maintained. The selected files are displayed in theSelected Files window. Click **OK**. The files are listedin a table on the wizard panel.
2. If you want to remove a file, select the file in the table and then click**Remove**.
3. Continue to add or remove files until you have the correct set offiles.
4. Click **Next**.

Specifying optional Web module properties

On the **Specifying Optional Web Module Properties** panel:

1. Indicate whether the module can be installed in a distributable Webcontainer. The default value is false.
2. Specify the full classpath for the Web application.
3. Click **Next**.

Choosing Web Module icons

On the **Choosing Web Module icons** panel, specify icons for yourmodule.

1. Specify the full path name of a GIF or JPEG file. The icon must be16x16 pixels in size.
2. Specify a full path name of a GIF or JPEG file. The icon must be32x32 pixels in size.
3. Click **Next**.

Adding Web components

On the **Adding Web components** panel, add new servlets or JSPfiles or import existing ones.

To add a new Web component:

1. Click **New**.
2. On the **Specifying Web Component Properties** panel, specify thecomponent name and enter values for other properties. View the help for6.6.8.0.1: Assembly properties for Web components.
3. On the **Specifying Web Component Type** panel, indicate the typeof Web component and specify the servlet class name or JSP file.
4. On the **Choosing Web Component Icons** panel, specify a filecontaining a JPEG or GIF image.
5. On the **Adding Security Role References** panel, enter values forsecurity role references. Click **Add** to enter a rolename. Click **OK**. The role name is displayed in thetable on the wizard panel. To remove a role, select the role in thetable and then click **Remove**. Repeat as necessary.View the help for 6.6.43.0.3: Assembly properties for security role references. Click **Next**.
6. On the **Adding Initialization Parameters** panel, enter values forthe Web component's initialization parameters. Click**Add** to add a parameter. You must enter a name andvalue. Click **OK**. The parameter is displayed in atable on the wizard panel. To remove a parameter, select the parameterand click **Remove**. Repeat as necessary. View the helpfor 6.6.8.0.2: Assembly properties for initialization parameters.

7. Click **Finish**.

To import an existing Web component:

1. Click **Import**.
2. Browse the file system to locate the desired archive. The contentsof the archive are displayed in a window. Select the desired componentand then click **Add**. The components are added to the SelectedComponents window. Click **OK**.

To remove a Web component, select the component name in the table and click**Remove**.

When you are finished adding Web components, click **Next**.

Adding security roles

On the **Adding Security Roles** panel:

1. Click **Add**. Type a role name and, optionally, type adescription. Click **OK**. The role name is displayed ina table on the wizard panel. View the help for 6.6.5.0.5: Assembly properties for security roles.
2. Continue to add security roles as needed. If you need to remove arole, select the role in the table and then click **Remove**.
3. Click **Next**.

Adding servlet mappings

On the **Adding Servlet Mappings** panel:

1. Click **Add**. Enter a URL pattern and select a servlet fromthe menu. View the help for 6.6.8.0.11: Assembly properties for servlet mapping. Click **OK**. The servlet mappings aredisplayed in a table on the wizard panel.
2. Continue to add and remove URL patterns and corresponding servlets asneeded. If you need to remove mapping, select the entry in the tableand then click **Remove**.
3. Click **Next**.

Adding resource references

On the **Adding Resource References** panel, enter references forresource connection factories.

1. Click **Add** to add a reference. You must enter a value fora name, type, and authorization mode. View the help for 6.6.43.0.2 Assembly properties for resource references. Click **OK**. The reference isdisplayed in the table on the wizard panel.
2. To remove a reference, select the reference in the table and then click**Remove**.
3. Continue to add and remove references as needed.
4. Click **Next**.

Adding context parameters

On the **Adding Context Parameters** panel, enter values for contextparameters.

1. Click **Add** to add a parameter. You must enter a name andvalue. View the help for 6.6.8.0.8: Assembly properties for context parameters. Click **OK**. The parameter isdisplayed in the table on the wizard panel.
2. To remove a parameter, select the parameter and then click**Remove**.

3. Continue to add and remove parameters as needed.
4. Click **Next**.

Adding error pages

On the **Adding Error Pages** panel, enter values for errorpages.

1. Click **Add** to add a page. You must enter alocation. Then choose **Error Code** or **ErrorException**. Enter a name for the error code or exception.View the help for 6.6.8.0.9: Assembly properties for error pages. Click **OK**. The error page isdisplayed in the table on the wizard panel.
2. To remove an error page, select the item in the table and then click**Remove**.
3. Continue to add and remove error pages as needed.
4. Click **Next**.

Adding MIME mappings

On the **Adding MIME Mappings** panel, enter values for MIMEmappings.

1. Click **Add** to add a mapping. You must enter an extensionand a MIME type. View the help for 6.6.8.0.10: Assembly properties for MIME mapping. Click **OK**. The mapping is displayedin the table on the wizard panel.
2. To remove a mapping, select the mapping and then click**Remove**.
3. Continue to add and remove mappings as needed.
4. Click **Next**.

Adding Tag Libraries

On the **Adding Tag Libraries** panel, enter values for taglibraries.

1. Click **Add** to add a tag library. You must enter a tagfile name and library location. View the help for 6.6.8.0.12: Assembly properties for tag libraries. Click **OK**. The tag library isdisplayed in the table on the wizard panel.
2. To remove a tag library, select the library and then click**Remove**.
3. Continue to add and remove tag libraries as needed.
4. Click **Next**.

Adding Welcome Files

On the **Adding Welcome Files** panel, enter values for welcomefiles.

1. Click **Add**. Enter a file name or use the file browser tolocate the file. View the help for 6.6.8.0.13: Assembly properties for welcome files. Click **OK**. The file name isdisplayed in the table on the wizard panel.
2. To remove a file, select the file in the table and then click**Remove**.
3. Continue to add and remove files as needed.
4. Click **Next**.

Adding EJB references

On the **Adding EJB References** panel, enter values for EJBreferences.

1. Click **Add** to add a reference. You must enter a value forthe name, home interface, remote interface, and type. View the help for6.6.43.0.1: Assembly properties for EJB references. Click **OK**. The reference isdisplayed in the table on the wizard panel.
2. To remove a reference, select the entry in the table and then click**Remove**.
3. Continue to add and remove references as needed.

Setting additional properties and saving the archive

Click **Finish** to complete the wizard. To change settingsfor properties, click **Back** to return to the appropriatepanel. Make any needed changes, and then click**Finish**.

After you click **Finish**, the contents of the archive aredisplayed in the Application Assembly Tool window. In the navigationpane, continue adding or modifying properties as needed. For example,you can add binding information. When you are finished editing thearchive, click **File**->**Save** to save the archivefile.