JSP files -- table of contents

Development

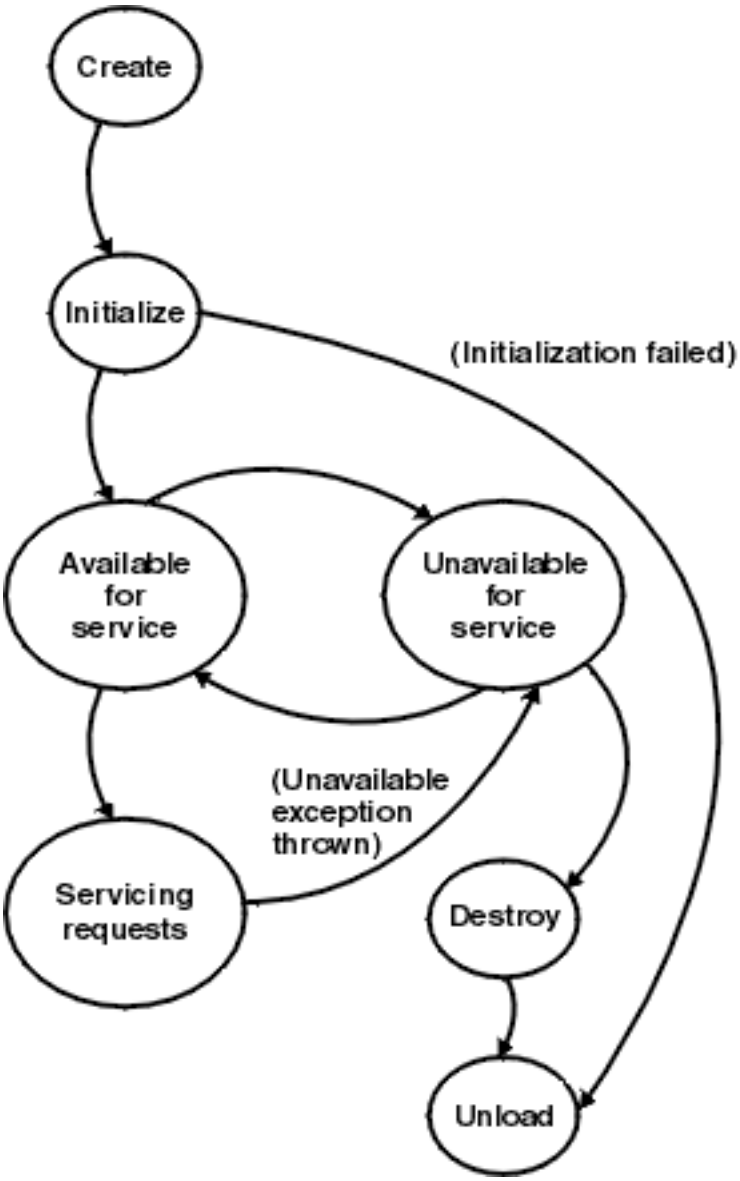
Administration

4.2.2: Developing JSP files

If JSP files are fairly new to you, consider reading about their lifecycle and access model. When you are ready to begin writing JSP files, see the article featuring JSP file content. Review the support and environment article for topics such as JSP processors and APIs, recommended development tools, and batch compiling.

4.2.2.1: JavaServer Pages (JSP) lifecycle

JSP files are compiled into servlets. After a JSP is compiled, its lifecycle is similar to the servlet lifecycle:



Java source generation and compilation

When a Web container receives a request for a JSP file, it passes the request to the JSP processor .

If this is the first time the JSP file has been requested or if the compiled copy of the JSP file is not found, the JSP compiler generates and compiles a Java source file for the JSP file. The JSP processor puts the Java source and class file in the JSP processor directory.

By default, the JSP syntax in a JSP file is converted to Java code that is added to the service() method of the generated class file. If you need to specify initialization parameters for the servlet or other initialization information, add the method directive set to the value `init`.

Request processing

After the JSP processor places the servlet class file in the JSP processor directory, the Web container creates an instance of the servlet and calls the servlet service() method in response to the request. All subsequent requests for the JSP are handled by that instance of the servlet.

When the Web container receives a request for a JSP file, the engine checks to determine whether the JSP file (.jsp) has changed since it was loaded. If it has changed, the Web container reloads the updated JSP file (that is, generates an updated Java source and class file for the JSP). The newly loaded servlet instance receives the client request.

Termination

When the Web container no longer needs the servlet or a newinstance of the servlet is being reloaded, the Web container invokes theservlet's destroy() method. The Web container can also call the destroy() method if the engine needs to conserve resources or a pending call to a servlet service() method exceeds the timeout. The Java Virtual Machine performs garbage collection after the destroy.

4.2.2.1a: JSP access models

JSP files can be accessed in two ways:

- The browser sends a request for a JSP file.

  The JSP file accesses beans or other components that generate dynamic content that is sent to the browser,as shown:

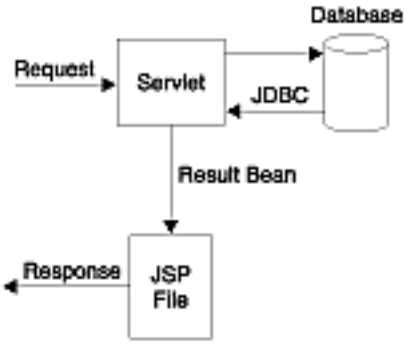
  Request for a JSP file

  When the Web server receives a request for a JSP file, the server sends therequest to the application server. The application server parses theJSP file and generates Java source, which is compiled and executed as aservlet.

- The request is sent to a servlet that generates dynamic content and calls a JSP file to send the content to the browser, as shown:

  Request for a servlet

  This access model facilitates separating content generation from content display.

  The application server supplies a set of methods in the HttpServiceRequest object and the HttpServiceResponse object. These methods allow an invoked servlet to place an object (usually a bean) into a request object and pass that request to another page (usually a JSP file) for display. The invoked page retrieves the beanfrom the request object and generates the client-side HTML.

4.2.2.2: JSP support and environment in WebSphere

IBM WebSphere Application Server supports the JSP 1.1 Specification from Sun Microsystems. If you are going to develop new JSP files for use with IBMWebSphere Application Server, it is recommended you use JSP 1.1.

All APIs described in this section are supported at the JSP 1.1 level.

4.2.2.2.2: JSP processors

When you install the Application Server product on a Web server, the Web server configuration is set to pass HTTP requests for JSP files (files with the extension .jsp) to the Application Server product.

The JSP processor creates and compiles a servlet from each JSP file. The processor produces these files for each JSP file:

- .java file, which contains the Java language code for the servlet
- .class file, which is the compiled servlet
- .dat file, which contains the static part of the original jsp file

The JSP processor puts the .java, the .class file, and the .dat file in the following path:

`<product_installation_root>\temp\<hostname>\<servername>\<webmodulename>`

Like all servlets, a servlet generated from a JSP file extends javax.servlet.http.HttpServlet. The servlet Java code contains import statements for the necessary classes and a package statement, if the servlet class is part of a package.

If the JSP file contains JSP syntax (such as directives and scriptlets), the JSP processor converts the JSP syntax to the equivalent Java code. If the JSP file contains HTML tags, the processor adds Java code so that the servlet outputs the HTML character by character.

4.2.2.2.3: Java Server Page attributes

Use the WebSphere Application Assembly Tool (AAT) to set the following Java Server Page attributes. The JSP attributes are storedin the IBM extensions document for Web module, `ibm-web-ext.xmi`.

JSP file attribute names	JSP file attribute values (Default values are in bold text)	Purpose
keepgenerated	true | **false**	If true, the generated `.java`file will be kept. If the value is false, the `.java` file isnot saved.
dosetattribute	true | **false**	By default, JSP files using the "usebean" tag withScope="session" do not always work properly when session persistence is enabled.
scratchdir	***product_installation_root*\temp**	Set `scratchdir` to a valid drive and directory which the JSP enginewill use to store the generated `.class` and `.java` files.
jsp.repeatTag.ignoreException	true | **false**	In previous releases, the <tsx:repeat> tagwould iterate until one of the following conditions was met: 1. The end value was reached 2. An `ArrayIndexOutofBoundsException` was thrown Other types of exceptions were caught but not thrown, which could result in numerous errors being returned to the browser. In version 4.0, the default behavior will now stop therepeat tag iterations when any exception is thrown. To reinstate the old behavior, set this parameter's valueto **true**.
defaultEncoding	Name of the desired character set. The value of the system property**file.encoding** is the default.	Use this parameter to set the encoding for JSP pages. If a JSP page contains a `contentType` directive that defines an alternative character set, that character set is used instead of the `defaultEncoding` parameter's value. The order of precedence is: 1. The JSP page's `contentType`directive's charset. 2. The `defaultEncoding` parameter's value. 3. The System property `file.encoding` value 4. ISO-8859-1

4.2.2.2.4: Batch Compiling JSP files

As an IBM enhancement to JSP support, IBM WebSphere Application Server provides a batch JSP compiler. Use this function to batch compile your JSP files and thereby enable faster responses to the initial client requests for the JSP files on your production Web server.

It is best to batch compile all of the JSP files associated with an application. Batch compiling saves system resources and provides security on the application server by specifying if and/or when the server is to check for a classfile or recompile a JSP file. The application server will monitor the compiled JSP file for changes, and will automatically recompile and reload the JSP file whenever the application server detects that the JSP file has changed. By modifying this process, you can eliminate time- and resource-consuming compilations and ensure that you have control over the compilation of your JSP files. It is also useful as a fast way to resynchronize all of the JSP files for an application.

4.2.2.2.4.2: Compiling JSP 1.1 files as a batch

To use the JSP batch compiler for JSP files, enter the following command on a single line at an operating system command prompt:

```
JspBatchCompiler -enterpriseApp<name>-webModule<name>[-filename<jsp name>]
    [-keepgenerated<true|false>][-configFile<configfile name>]
```

> **Note:** If the names specified for these arguments are comprised of two or more words separated by spaces, you must add quotation marks around the names.

where:

- **enterpriseApp**

  The name of the Enterprise Application you want to compile.

- **webModule**

  The name of the specific Web module that you want to compile.

- **filename**

  The name of a single JSP file that you want to compile. If this argument is not set, all files in the Web module are compiled.

- **keepgenerated**

  If set to "yes" WebSphere Application Server will save the generated ***.java*** files used for compilation on your server. By default, this is set to "no" and the ***.java*** files are erased after the class files have been compiled.

- **configFile**

  The `configFile` parameter is valid only on Advanced Single Server Edition for "Multiplatforms." Use it to specify an alternative sever configuration file (the default is `server-cfg.xml`).

4.2.2.3: Overview of JSP file content

JSP files have the extension .jsp. A JSP filecontains any combination of the following items. Click an item to learn about its syntax. To learn how to put it all together, see the Related information for examples, samples, and additional syntax references.

JSP syntax

Syntax format	Details
Directives	Use JSP directives (enclosed within <%@ and %>) to specify: • Scripting language being used • Interfaces a servlet implements • Classes a servlet extends • Packages a servlet imports • MIME type of the generated response  See Sun's JSP Syntax Referencefor JSP 1.1 syntax descriptions and examples.
Class-wide variable and method declarations	Use the <%! *declaration(s)* %> syntax to declareclass-wide variables and class-wide methods for the servlet class.
Inline Java code (scriptlets), enclosed within <% and %>	You can embed any valid Java language code inline withina JSP file between the <% and %> tags. Suchembedded code is called a *scriptlet*. If you do not specify the method directive, the generated code becomes the body of the service method. An advantage of embedding Java coding inline in JSP files is that the servlet does not have to be compiled in advance, and placed on the server. Thismakes it easier to quickly test servlet coding.
Variable text, specified using IBM extensions for variable data or Java expressions enclosed within <%= and %>	The IBM extensions are the more user-friendly approach to putting variable fields on your HTML pages. A second method for adding variable data is to specify a Java language expression that is resolved when the JSP file is processed. Use the JSP expression tags <%= and %>. The expression is evaluated, converted into a string, and displayed. Primitive types, such as int and float, areautomatically converted to string representation.
<jsp:useBean> tag	Use the <jsp:useBean> tag to create an instance of a bean that will be accessed elsewhere within the JSP file. Then use JSP tags to access the bean.
JSP tags for database access (JSP 1.1)	The IBM extensions make it easy for non-programmers to create Web pages that access databases.

HTML tags

A JSP file can contain any valid HTML tags. View article 0.70: What is HTML? for more informationon HTML. Refer to your favorite HTMLreference for a description of HTML tags.

4.2.2.3.2: JSP syntax: Class-wide variables and methods

Use the <%! *declaration(s)* %> syntax to declareclass-wide variables and class-wide methods for the servlet class.

An example of specifying class-widevariables and methods:

```
<%! int i=0; String foo = "Hello"; %>   <%! private void foo() {// code for the method } %>
```

4.2.2.3.3: JSP syntax: Inline Java code (scriptlets)

You can embed any valid Java language code inlinebetween the <% and %> tags. Suchembedded code is called a *scriptlet*. If you do not specify the method directive, the generated code becomes the body of the service method.

Be sure to use the braces characters, { }, to enclose if, while, and for statements even if the scope contains a single statement. You can enclose the entire statement with a single scriptlet tag. However, if you use multiple scriptlet tags with the statement, be sure to place the opening brace character, {, in the same statement as the if, while, or for keyword. The following examples illustrate these points. The first example is the easiest.

```
<%for (int i = 0; i < 1; i++)   {   out.println("<P>This is written when " + i + " is < 1</P>");
}%>...<% for (int i = 0; i < 1; i++) {                              %><%
out.println("<P>This is written when " + i + " is < 1</P>"); %><% }
%>...<% for (int i = 0; i < 1; i++)      {
%><%    out.println("<P>This is written when " + i + " is < 1</P>"); %><%    }
%>
```

4.2.2.3.4: JSP syntax: Java expressions

To specify a Java language expression that is resolvedwhen the JSP file is processed, use the JSP expression tags <%= and%>. The expression is evaluated, converted into a string,and displayed. Primitive types, such as int and float, areautomatically converted to string representation. In this example, foois the class-wide variable declared in the class-wide variables and methods example:

```
<p>Translate the greeting <%= foo %>.</p>
```

When the JSP file is served, the text reads: `Translate the greeting Hello.`

4.2.2.3.5: JSP syntax: useBean tag

The `<jsp:useBean>` tag locates a Bean or creates an instance of a Bean if it does not exist.

JavaBeans can be class files, serializedbeans, or dynamically generated by a servlet.A JavaBean can even be a servlet (that is, provide a service). If aservlet generates dynamic content and stores it in a bean, the bean can thenbe passed to a JSP file for use within the Web page defined by thefile.

See Sun's JSP Syntax Referencefor JSP 1.1 syntax descriptions and examples.

4.2.2.3.5.1: JSP syntax: <jsp:useBean> tag

Use the <jsp:useBean> tag to locate or instantiate a JavaBeans component. The syntax for the <jsp:useBean> tag is:

```
<jsp:useBean
  id="beanSomeName"
 scope="page|request|session|applicaton"
{ class="package_class" |
 type ="package_class" |
 class="package_class" type ="package_class" |
 beanName="{package.class| <%= expression%>}" type ="package_class"
}
{ />|
 > other elements
 </jsp:useBean>
}
```

See Sun's JSP syntax referencefor a description of the <jsp:useBean> attributes and examples.

4.2.2.3.5.2: JSP syntax: Accessing bean properties

After specifying the `<jsp:useBean>` tag, you can access the bean at any pointwithin the JSP file using the `<jsp:getProperty>` tag.

For a description of the `<jsp:getProperty>` tag attributesand for coding examples, see Sun's JSP Syntax Reference

4.2.2.3.5.3: JSP syntax: Setting bean properties

You can set bean properties by using the `<jsp:setProperty>` tag. The `<jsp:setProperty>` tag specifies a list of properties and the corresponding values. The properties areset after the the bean is instantiated using the `<jsp:useBean>` tag.

You must declare the bean with `<jsp:useBean>` before you can set a property value.

See the Sun's JSP syntax referencefor `<jsp:setProperty>` syntax details and examples.

4.2.2.3.7: IBM extensions to JSP syntax

Refer to the Sun JSP Specification for the base JavaServer Pages (JSP) APIs. IBMWebSphere Application Server Version 3.5 provided several extensions to the base APIs.The backward compatibility of the JSP 1.1 specification to JSP 1.0 allows users to invoke these APIs without modification.

The extensions belong to these categories:

Extension	Use
Syntax for variable data	Put variable fields in JSP files and have servlets and JavaBeans dynamicallyreplace the variables with values from a database when the JSP output is returned tothe browser
Syntax for database access	Add a database connection to a Web page and then use that connection to query or updatethe database. The user ID and password for the database connection can be provided by theuser at request time, or can be hardcoded within the JSP file.

Scope of variables: Because the values specified by syntax apply onlyto the JSP file in which thesyntax is embedded, identifiers and other tag data can be accessed only withinthe page.

See the Related information for syntax details.

4.2.2.3.7.1: JSP syntax: Tags for variable data

The variable data syntax enables you to put variable fields in your JSP file and have your servlets and JavaBeansdynamically replace the variables with values from a database when the JSP output is returned to the browser.

The table summarizes the tags. Click a tag to link to its syntax description.

Goal	Tag	Details
Get the value of a bean to display in a JSP.	<tsx:getProperty>	This IBM extension of the Sun JSP <jsp:getProperty> tag implements all of the <jsp:getProperty> function and adds the ability to introspect a database bean that was created using the IBM extension <tsx:dbquery> or <tsx:dbmodify>. **Note:** You cannot assign the value from this tag toa variable. The value, generated as output from this tag, is displayed in the Browser window.
Repeat a block of HTML tagging that contains the <tsx:getProperty> syntax and the HTML tags for formatting content.	<tsx:repeat>	Use the <tsx:repeat> syntax to iterate over a database query results set. The <tsx:repeat> syntax iterates from the start value to the end value until one of the following conditions is met: ● The end value is reached. ● An exception is thrown. The output of a <tsx:repeat> block is buffered until the block completes. If an exception is thrown before a block completes, no output is written for that block.

4.2.2.3.7.1.1: JSP syntax: <tsx:getProperty> tag syntax and examples

```
<tsx:getProperty name="bean_name"  property="property_name" />
```

where:

- **name**

  The name of the JavaBean declared by the `id` attribute of a <tsx:dbquery> syntax within the JSP file. See <tsx:dbquery> for an explanation. The value of this attribute is case-sensitive.

- **property**

  The property of the bean to access for substitution. The value ofthe attribute is case-sensitive and is the locale-independent name of theproperty.

Examples

```
<tsx:getProperty name="userProfile" property="username" /><tsx:getProperty name="request"
property=request.getParameter("corporation") />
```

In most cases, the value of the property attribute will be just theproperty name. However, to access the request bean or access a property of a property(sub-property), you specify the full form of the property attribute.The full form also gives you the option to specify an index for indexedproperties. The optional index can be a constant (such as 2) or anindex like the one described in <tsx:repeat>. Some examples of using the full form of the property attribute:

```
<tsx:getProperty name="staffQuery" property=address(currentAddressIndex) /><tsx:getProperty
name="shoppingCart" property=items(4).price /><tsx:getProperty name="fooBean"
property=foo(2).bat(3).boo.far />
```

4.2.2.3.7.1.2: JSP syntax: <tsx:repeat> tag syntax

`<tsx:repeat index=`*`name`* `start="`*`starting_index`*`" end="`*`ending_index`*`"></tsx:repeat>`

where:

- **index**

  An optional name used to identify the index of this repeat block.The value is case-sensitive and its scope is the JSP file.

- **start**

  An optional starting index value for this repeat block. The defaultis 0.

- **end**

  An optional ending index value for this repeat block. The maximumvalue is 2,147,483,647. If the value of the **end** attribute is less thanthe value of the **start** attribute, the **end** attribute is ignored.

4.2.2.3.7.1.2a: JSP syntax: The repeat tag results set and the associated bean

The <tsx:repeat> iterates over a results set. The results set is contained within a JavaBean. The bean can be a static bean (for example, a bean created by using the IBM WebSphere Studio database wizard) or a dynamically generated bean (for example, a bean generated by the <tsx:dbquery> syntax). The following table is a graphic representation of the contents of a bean, myBean:

	col1	col2	col3
row0	friends	Romans	countrymen
row1	bacon	lettuce	tomato
row2	May	June	July

Some observations about the bean:

- The column names in the database table become the property names of the bean. The section <tsx:dbquery> describes a technique for mapping the column names to different property names.
- The bean properties are indexed. For example, myBean.get(Col1(row2)) returns `May`.
- The query results are in the rows. The <tsx:repeat> iterates over the rows (beginning at the start row).

The following table compares using the <tsx:repeat> to iterate over static bean versus a dynamically generated bean:

Static Bean Example	**<tsx:repeat> Bean Example**

myBean.class
`// Code to get a connection// Code to get the data  Select * from myTable;// Code to close the connection`
JSP file
`<tsx:repeat index=abc>  <tsx:getPropery name="myBean"  property="col1(abc)" /></tsx:repeat>`
ⓘ
• The bean (myBean.class) is a static bean.
• The method to access the bean properties is myBean.get(*property*(*index*)).
• You can omit the property index, in which case the index of the enclosing <tsx:repeat> is used. You can also omit the index on the <tsx:repeat>.
• The <tsx:repeat> iterates over the bean properties row by row, beginning with the start row.
	JSP file
	`<tsx:dbconnect id="conn"userid="alice"passwd="test"url="jdbc:db2:sample"driver="COM.ibm.db2.jdbc.app.DB2Dri`
	`><tsx:dbquery id="dynamic" connection="conn" >  Select * from myTable;</tsx:dbquery><tsx:repeat index=abc>`
	`name="dynamic"  property="col1(abc)" /></tsx:repeat>`
	ⓘ
	• The bean (dynamic) is generated by the <tsx:dbquery> and does not exist until the syntax is executed.
	• The method to access the bean properties is dynamic.getValue("*property*", *index*).
	• You can omit the property index, in which case the index of the enclosing <tsx:repeat> is used. You can also omit the index on the <tsx:repeat>.
	• The <tsx:repeat> syntax iterates over the bean properties row by row, beginning with the start row.

Implicit and explicit indexing

Examples 1, 2, and 3 show how to use the <tsx:repeat>. Theexamples produce the same output if all indexed properties have 300 or fewerelements. If there are more than 300 elements, Examples 1 and 2 willdisplay all elements, while Example 3 will show only the first 300elements.

Example 1 shows implicit indexing with the default start and default endindex. The bean with the smallest number of indexed properties restricts the number of times the loop will repeat.

```
<table><tsx:repeat>  <tr><td><tsx:getProperty name="serviceLocationsQuery" property="city" /></tr></td>  <tr><td><tsx:getProperty
name="serviceLocationsQuery" property="address" /></tr></td>  <tr><td><tsx:getProperty name="serviceLocationsQuery"
property="telephone" /></tr></td></tsx:repeat></table>
```

Example 2 shows indexing, starting index, and ending index:

```
<table><tsx:repeat index=myIndex start=0 end=2147483647>  <tr><td><tsx:getProperty name="serviceLocationsQuery"
property=city(myIndex) /></tr></td>  <tr><td><tsx:getProperty name="serviceLocationsQuery" property=address(myIndex) /></tr></td>
<tr><td><tsx:getProperty name="serviceLocationsQuery" property=telephone(myIndex) /></tr></td></tsx:repeat></table>
```

Example 3 shows explicit indexing and ending index with implicit startingindex. Although the index attribute is specified, the indexed propertycity can still be implicitly indexed because the `(myIndex)` is not required.

```
<table><tsx:repeat index=myIndex end=299>  <tr><td><tsx:getProperty name="serviceLocationsQuery" property="city" /t></tr></td>
<tr><td><tsx:getProperty name="serviceLocationsQuery" property="address(myIndex)" /></tr></td>  <tr><td><tsx:getProperty
name="serviceLocationsQuery" property="telephone(myIndex)" /></tr></td></tsx:repeat></table>
```

Nesting <tsx:repeat> blocks

You can nest <tsx:repeat> blocks. Each block is separatelyindexed. This capability is useful for interleaving properties on twobeans, or properties that have sub-properties. In the example, two<tsx:repeat> blocks are nested to display the list of songs on each compactdisc in the user's shopping cart.

```
<tsx:repeat index=cdindex>  <h1><tsx:getProperty name="shoppingCart" property=cds.title /></h1>  <table>  <tsx:repeat>
<tr><td><tsx:getProperty name="shoppingCart" property=cds(cdindex).playlist />    </td></tr>  </table>
</tsx:repeat></tsx:repeat>
```

4.2.2.3.7.2: JSP syntax: Tags for database access

Beginning with IBM WebSphere Application Server Version 3.x, the JSP 1.0 supportwas extended to provide syntaxfor database access. The syntax makes it simple to add a database connectionto a Web page and then use that connection to query or update the database.The user ID and password for the database connection can be provided by theuser at request-time or hard coded within the JSP file.

The table summarizes the tags. Click a tag to link to its syntax description.

Goal	Tag	Details and examples
Specify information needed to make a connection to a JDBC or an ODBC database.	<tsx:dbconnect>	The <tsx:dbconnect> syntax does not establish the connection. Instead,the <tsx:dbquery> and <tsx:dbmodify> syntax are used to referencea <tsx:dbconnect> in the same JSP file and establish the connection. When the JSP file is compiled into a servlet, the Java processor addsthe Java coding for the <tsx:dbconnect> syntax to the servlet'sservice() method, which means a new database connection is created for eachrequest for the JSP file.
Avoid hard coding the user ID and password in the<tsx:dbconnect>.	<tsx:userid> and <tsx:passwd>	Use the <tsx:userid> and <tsx:passwd> to acceptuser input for the values and then add that data to the request object. The request objectcan be accessed by a JSP file (such as the JSPEmployee.jsp example) that requests the databaseconnection. The <tsx:userid> and <tsx:passwd> must be used within a<tsx:dbconnect> tag.
Establish a connection to a database, submit database queries, and return the results set.	<tsx:dbquery>	The <tsx:dbquery>: 1. References a <tsx:dbconnect> in the same JSP file and uses the information it provides to determine the database URL and driver. The user ID and password are also obtained from the <tsx:dbconnect> if those values are provided in the <tsx:dbconnect>. 2. Establishes a new connection 3. Retrieves and caches data in the results object 4. Closes the connection (releases the connection resource)

Establish a connection to a database and then add records to a database table.	<tsx:dbmodify>	The <tsx:dbmodify>: 1. References a <tsx:dbconnect> in the same JSP file and uses the information provided by that to determine the database URL and driver. The user ID and password are also obtained from the <tsx:dbconnect> if those values are provided in the <tsx:dbconnect>. 2. Establishes a new connection 3. Updates a table in the database 4. Closes the connection (releases the connection resource) **Examples:** Basic example
Display query results.	<tsx:repeat> and <tsx:getProperty>	The <tsx:repeat> loops through each of the rows in the query results.The <tsx:getProperty> uses the query results object (for the <tsx:dbquery>syntax whose identifier is specified by the <tsx:getProperty> bean attribute)and the appropriate column name (specified by the <tsx:getProperty> propertyattribute) to retrieve the value. **Note:** You cannot assign the value from the <tsx:getProperty> tag toa variable. The value, generated as output from this tag, is displayed in the Browser window. **Examples:** Basic example

4.2.2.3.7.2.1: JSP syntax: <tsx:dbconnect> tag syntax

```
<tsx:dbconnect id="connection_id"       userid="db_user" passwd="user_password"
url="jdbc:subprotocol:database" driver="database_driver_name"
jndiname="JNDI_context/logical_name"></tsx:dbconnect>
```

where:

- **id**

  A required identifier. The scope is the JSP file. This identifier is referenced by the connection attribute of a <tsx:dbquery> tag.

- **userid**

  An optional attribute that specifies a valid user ID for the database to be accessed. If specified, this attribute and its value are added to the request object.

  Although the userid attribute is optional, the userid must be provided. See <tsx:userid> and <tsx:passwd> for an alternative to hard coding this information in the JSP file.

- **passwd**

  An optional attribute that specifies the user password for the userid attribute. (This attribute is not optional if the userid attribute is specified.) If specified, this attribute and its value are added to the request object.

  Although the passwd attribute is optional, the password must be provided. See <tsx:userid> and <tsx:passwd> for an alternative to hard coding this attribute in the JSP file.

- **url** and **driver**

  To establish a database connection, the URL and driver must be provided.

  The Application Server Version 3 supports connection to JDBC databases and ODBC databases.

JDBC database

For a JDBC database, the URL consists of the following colon-separated elements: jdbc, the sub-protocol name, and the name of the database to be accessed. An example for a connection to the Sample database included with IBM DB2 is:

```
url="jdbc:db2:sample"driver="COM.ibm.db2.jdbc.app.DB2Driver"
```

ODBC database

Use the Sun JDBC-to-ODBC bridge driver included in the Java Development Kit (JDK) oranother vendor's ODBC driver.

The url attribute specifies the location of the database. The driver attribute specifies the name of the driver to be used to establish the database connection.

If the database is an ODBC database, you can use an ODBC driver or the Sun JDBC-to-ODBC bridge included with the JDK. If you want to use an ODBC driver, refer to the driver documentation for instructions on specifying the database location (the url attribute) and the driver name.

In the case of the bridge, the url syntax is jdbc:odbc:*database*. An example is:

```
url="jdbc:odbc:autos"driver="sun.jdbc.odbc.JdbcOdbcDriver"
```


To enable the Application Server to access the ODBC database, use the ODBC Data Source Administrator to add the ODBC data source to the System DSN configuration. To access the

ODBC Administrator, click the ODBC icon on the Windows NT Control Panel.

- **jndiname**

  An optional attribute that identifies a valid context in the Application Server JNDI naming context and the logical name of the data source in that context. The context is configured by the Web administrator using an administrative client such as the WebSphere Administrative Console.

  If the jndiname is specified, the JSP processor ignores the driver and url attributes on the <tsx:dbconnect> tag.

An empty element (such as <url></url>) is valid.

4.2.2.3.7.2.2: JSP syntax: <tsx:userid> and <tsx:passwd> tag syntax

```
<tsx:dbconnect id="connection_id"       <font color="red"><userid></font><tsx:getProperty
name="request" property=request.getParameter("userid") /><font color="red"></userid></font>   <font
color="red"><passwd></font><tsx:getProperty name="request" property=request.getParameter("passwd")
/><font color="red"></passwd></font>   url="protocol:database_name:database_table"
driver="JDBC_driver_name"> </tsx:dbconnect>
```

where:

- **<tsx:getProperty>**

  This syntax is a mechanism for embedding variable data. See JSP syntax for variable data.

- **userid**

  This is a reference to the request parameter that contains the userid. The parameter must have already been added to the request object that was passed to this JSP file. The attribute and its value can be set in the request object using an HTML form or a URL query string to pass the user-specified request parameters.

- **passwd**

  This is a reference to the request parameter that contains the password. The parameter must have already been added to the request object that was passed to this JSP. The attribute and its value can be set in the request object using an HTML form or a URL query string to pass user-specified values.

4.2.2.3.7.2.3: JSP syntax: <tsx:dbquery> tag syntax

```
<%-- SELECT commands and (optional) JSP syntax can be placed within the tsx:dbquery. --%><%-- Any other syntax, including HTML comments, are not valid. --%><tsx:dbquery id="query_id" connection="connection_id" limit="value" ></tsx:dbquery>
```

where:

- **id**

  The identifier of this query. The scope is the JSP file. This identifier is used to reference the query, for example, from the <tsx:getProperty> to display query results.

  The id becomes the name of a bean that contains the results set. The bean properties are dynamic and the property names are the names of the columns in the results set. If you want different column names, use the SQL keyword for specifying an alias on the SELECT command. In the following example, the database table contains columns named FNAME and LNAME, but the SELECT statement uses the AS keyword to map those column names to FirstName and LastName in the results set:

  ```
  Select FNAME, LNAME AS FirstName, LastName from Employee where FNAME='Jim'
  ```

- **connection**

  The identifier of a <tsx:dbconnect> in this JSP file. That <tsx:dbconnect> provides the database URL, driver name, and (optionally) the user ID and password for the connection.

- **limit**

  An optional attribute that constrains the maximum number of records returned by a query. If the attribute is not specified, no limit is used. In such a case, the effective limit is determined by the number of records and the system caching capability.

- SELECT command and JSP syntax

  The only valid SQL command is SELECT because the <tsx:dbquery> must return a results set. Refer to your database documentation for information about the SELECT command. See other sections of this document for a description of JSP syntax for variable data and inline Java code.

4.2.2.3.7.2.3a: Example: JSP syntax: <tsx:dbquery> tag syntax

In the following example, a database is queried for data about employees in a specified department. The department is specified using the <tsx:getProperty> to embed a variable data field. The value of the field is based on user input.

```
<tsx:dbquery id="empqs" connection="conn" >select * from Employee where WORKDEPT='<tsx:getProperty
name="request" property=request.getParameter("WORKDEPT") />'</tsx:dbquery>
```

4.2.2.3.7.2.4: JSP syntax: <tsx:dbmodify> tag syntax

```
<%-- Any valid database update commands can be placed within the DBMODIFY tag. --><%-- Any other
syntax, including HTML comments, are not valid. --><tsx:dbmodify
connection="connection_id"></tsx:dbmodify>
```

where:

- **connection**

  The identifier of a <DBCONNECT> tag in this JSP file. The <DBCONNECT> tag provides the database URL, driver name, and (optionally) the user ID and password for the connection.

- Database commands

  Valid database commands. Refer to your database documentation for details

4.2.2.3.7.2.4a: Example: JSP syntax: <tsx:dbmodify> tag syntax

In the following example, a new employee record is added to a database. The values of the fields are based on user input from this JSP and referenced in the database commands using <tsx:getProperty>.

```
<tsx:dbmodify connection="conn" >insert into EMPLOYEE
(EMPNO,FIRSTNME,MIDINIT,LASTNAME,WORKDEPT,EDLEVEL)values('<tsx:getProperty name="request"
property=request.getParameter("EMPNO") />','<tsx:getProperty name="request"
property=request.getParameter("FIRSTNME") />','<tsx:getProperty name="request"
property=request.getParameter("MIDINIT") />','<tsx:getProperty name="request"
property=request.getParameter("LASTNAME") />','<tsx:getProperty name="request"
property=request.getParameter("WORKDEPT") />',<tsx:getProperty name="request"
property=request.getParameter("EDLEVEL") />)</tsx:dbmodify>
```

4.2.2.3.7.2.5a: Example: JSP syntax: <tsx:repeat> and <tsx:getProperty> tags

```
<tsx:repeat><tr>        <td><tsx:getProperty name="empqs" property="EMPNO" />   <tsx:getProperty
name="empqs" property="FIRSTNME" />    <tsx:getProperty name="empqs" property="WORKDEPT" />
<tsx:getProperty name="empqs" property="EDLEVEL" />     </td></tr></tsx:repeat>
```

4.2.2.3a: JSP examples

The example JSP application accesses the Sample database that you can install with IBM DB2. The example application includes:

JSPLogin.jsp	An interface for logging in to the application
JSPEmployee.jsp	A dialog for querying and updating database records
JSPEmployeeRepeatResults.jsp	A dialog for displaying update confirmations and query results

JSP code example - a login

```
<HTML><HEAD><TITLE>JSP:  Login into the Employee Records
Center</TITLE></HEAD><BODY><H1><CENTER>Login into the Employee Records Center</CENTER></H1><FORM
NAME="LoginForm" ACTION="jsp10employee.jsp" METHOD="post"
ENCODE="application/x-www-form-urlencoded"><P>To login to the Employee Records Center, submit a
validuserid and password to access the Sample database installed under IBM DB2.</P><TABLE><TR
VALIGN=TOP ALIGN=LEFT><TD><B><I>Userid:</I></B></TD><TD><INPUT TYPE="text" NAME="USERID"
VALUE="userid"><BR></TD></TR><TR VALIGN=TOP ALIGN=LEFT><TD><B><I>Password:</I></B></TD><TD><INPUT
TYPE="password" NAME="PASSWD" VALUE="password"></TD></TR></TABLE><INPUT TYPE="submit" NAME="Submit"
VALUE="LOGIN"></FORM><HR></BODY></HTML>
```

JSP code example - view employee records

```
<HTML><HEAD><TITLE>JSP:  Add and View Employee Records</TITLE></HEAD><BODY><H1><CENTER>Add and View
Employee Records</CENTER></H1><%-- Get a connection to the Sample DB2 database using parameters from
Login.jsp --%><tsx:dbconnect id="conn"  url="jdbc:db2:sample"
driver="COM.ibm.db2.jdbc.app.DB2Driver"><userid><tsx:getProperty name="request" property="USERID"
/></userid><passwd><tsx:getProperty name="request" property="PASSWD"
/></passwd></tsx:dbconnect><FORM NAME="EmployeeForm" ACTION="employeeRepeatResults.jsp"
METHOD="post" ENCODE="application/x-www-form-urlencoded"><h2>Add Employee Record</h2><P>To add a new
employee record to the database, submit the following data:</P><TABLE><TR VALIGN="TOP"
ALIGN="LEFT"><TD><B><I>Employee Number:<br>(1 to 6 characters)</I></B></TD><TD> <INPUT TYPE="text"
NAME="EMPNO"> </TD></TR><TR VALIGN="TOP" ALIGN="LEFT"><TD><B><I>First name:</I></B></TD><TD><INPUT
TYPE="text" NAME="FIRSTNME" VALUE="First Name"><BR></TD></TR><TR VALIGN="TOP"
ALIGN="LEFT"><TD><B><I>Middle Initial:</I></B></TD><TD><INPUT TYPE="text" NAME="MIDINIT"
VALUE="M"><BR></TD></TR><TR VALIGN="TOP" ALIGN="LEFT"><TD><B><I>Last Name: </I></B></TD><TD><INPUT
TYPE="text" NAME="LASTNAME" VALUE="Last Name"><BR></TD></TR><TR VALIGN="TOP" ALIGN="LEFT"><TD><%--
Query the database to get the list of departments --%><tsx:dbquery id="qs" connection="conn" >
select * from DEPARTMENT  </tsx:dbquery><B><I>Department:</I></B></TD><TD><SELECT NAME="WORKDEPT"
><tsx:repeat> <OPTION VALUE= "<tsx:getProperty name="qs" property="DEPTNO" />" ><tsx:getProperty
name="qs" property="DEPTNAME" /></tsx:repeat></SELECT></TD></TR><TR VALIGN="TOP"
ALIGN="LEFT"><TD><B><I>Education:</I></B></TD><TD><SELECT NAME="EDLEVEL"><OPTION VALUE="1"
SELECTED>BS<OPTION VALUE="2">MS<OPTION VALUE="3">PhD</SELECT></TD></TR></TABLE><INPUT TYPE="submit"
NAME="Submit" VALUE="Update"><INPUT TYPE="hidden" NAME="USERID" VALUE="<tsx:getProperty
name="request" property="USERID" />"><INPUT TYPE="hidden" NAME="PASSWD" VALUE="<tsx:getProperty
name="request" property="PASSWD" />"></FORM><HR><FORM NAME="EmployeeForm"
ACTION="jsp10employeeRepeatResults.jsp" METHOD="post"
ENCODE="application/x-www-form-urlencoded"><h2>View Employees by Department</h2><P>To view records
for employees by department, select the departmentand submit the query:</P><TABLE><TR VALIGN="TOP"
ALIGN="LEFT"><TD><B><I>Department:</I></B></TD><TD><%-- Use the bean generated by earlier QUERY tag
--%><SELECT NAME="WORKDEPT" ><tsx:repeat> <OPTION VALUE= "<tsx:getProperty name="qs"
property="DEPTNO" />" ><tsx:getProperty name="qs" property="DEPTNAME"
/></tsx:repeat></SELECT></TD></TR></TABLE><INPUT TYPE="submit" NAME="Submit" VALUE="Query"><INPUT
TYPE="hidden" NAME="USERID" VALUE="<tsx:getProperty name="request" property="USERID" />"><INPUT
TYPE="hidden" NAME="PASSWD" VALUE="<tsx:getProperty name="request"  property="PASSWD"
/>"></FORM><HR></BODY></HTML>
```

JSP code example - EmployeeRepeatResults

```
<HTML><HEAD><TITLE>JSP Employee Results</TITLE></HEAD><H1><CENTER>EMPLOYEE RESULTS</CENTER></H1><BODY><tsx:dbconnect id="conn"
url="jdbc:db2:sample" driver="COM.ibm.db2.jdbc.app.DB2Driver"><userid><tsx:getProperty name="request"
property=request.getParameter("USERID") /></userid><passwd><tsx:getProperty name="request"
property=request.getParameter("PASSWD") /></passwd></tsx:dbconnect><% if  ( ( request.getParameter("Submit")).equals("Update") )
{ %><tsx:dbmodify connection="conn" >   INSERT INTO  EMPLOYEE    (EMPNO,FIRSTNME,MIDINIT,LASTNAME,WORKDEPT,EDLEVEL)   VALUES    (
'<tsx:getProperty name="request" property=request.getParameter("EMPNO") />',     '<tsx:getProperty name="request"
property=request.getParameter("FIRSTNME") />',     '<tsx:getProperty name="request" property=request.getParameter("MIDINIT") />',
'<tsx:getProperty name="request" property=request.getParameter("LASTNAME") />',    '<tsx:getProperty name="request"
property=request.getParameter("WORKDEPT") />',    <tsx:getProperty name="request" property=request.getParameter("EDLEVEL") />)
</tsx:dbmodify> <B><UL>UPDATE SUCCESSFUL</UL></B> <BR><BR><tsx:dbquery id="qs" connection="conn" >   select * from Employee
where  WORKDEPT= '<tsx:getProperty name="request" property=request.getParameter("WORKDEPT")
/>'</tsx:dbquery><B><CENTER><U>EMPLOYEE LIST</U></CENTER></B><BR><BR><HR><TABLE><TR
VALIGN=BOTTOM><TD><B>EMPLOYEE<BR><U>NUMBER</U></B></TD><TD><B><U>NAME</U></B></TD><TD><B><U>DEPARTMENT</U></B></TD>
<TD><B><U>EDUCATION</U></B></TD></TR><tsx:repeat><TR><TD><B><I><tsx:getProperty name="qs" property="EMPNO"
/></I></B></TD><TD><B><I><tsx:getProperty name="qs" property="FIRSTNME" /></I></B></TD><TD><B><I><tsx:getProperty name="qs"
property="WORKDEPT" /></I></B></TD><TD><B><I><tsx:getProperty name="qs" property="EDLEVEL" /></I></B></TD></TR></tsx:repeat>
</TABLE><HR><BR><% } %><% if  ( ( request.getParameter("Submit")).equals("Query") ) { %><tsx:dbquery id="qs2" connection="conn" >
select * from Employee  where  WORKDEPT= '<tsx:getProperty name="request" property=request.getParameter("WORKDEPT")
/>'</tsx:dbquery><B><CENTER><U>EMPLOYEE LIST</U></CENTER></B><BR><BR><HR><TABLE><TR><TR
VALIGN=BOTTOM><TD><B>EMPLOYEE<BR><U>NUMBER</U></B></TD><TD><B><U>NAME</U></B></TD><TD><B><U>DEPARTMENT</U></B></TD><TD><B><U>EDUCATION</U></B></TD></TR><tsx:repeat><TR><TD><B><I><tsx:getProperty
name="qs2" property="EMPNO" /></I></B></TD><TD><B><I><tsx:getProperty name="qs2" property="FIRSTNME"
/></I></B></TD><TD><B><I><tsx:getProperty name="qs2" property="WORKDEPT" /></I></B></TD><TD><B><I><tsx:getProperty name="qs2"
property="EDLEVEL" /></I></B></TD></TR></tsx:repeat> </TABLE><HR><BR><% } %></BODY></HTML>
```

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

 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

6.6.7.3.4: Updating Web container configurations with the Web console

During this task, you will update the configuration of anexisting Web container, which is part of an application serverconfiguration.

To update a Web container configuration:

1. Click **Nodes** -> ***hostname*** -> **Application Servers** -> ***application_server_name*** -> **Web Container** where *application_server_name* is the name of the existing application server.
2. Click Web Container. Its properties will be displayed on the rightside of the console.
3. Modify the properties.
4. When you are finished, click **OK**.
5. Save your configuration.
6. (Optional) To have the configuration take effect:
    1. Stop the server
    2. Start the server again.

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.

6.6.8.3: Administering Web modules with the Web console

Use the Web console to edit the configurations of Web modules. Because Web modules are configured, added, and removed as part of installed applications (.ear files), most of their settings displayed in this console are read-only.

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

Nodes -> ***hostname*** -> **Application Servers**
-> ***application_server_name*** -> **Web Containers**
-> **Installed Web modules**

6.6.8.3.1: Precompiling JSP files for Web modules of an application with the Web console

You can precompile the JSP files in a Web module either while youare installing the Web module (or the application containing it), orafter installation.

To precompile the JSP files during application installation, follow the instructions for installing an application.

To precompile the JSP files of an already installed application,follow the instructions for mapping virtual hosts to Web modules task.

In either case, you will end up at the "Mapping virtual hosts to Web modules" panelof the application installation wizard, from which you can specify to precompile JSPfiles.

6.6.8.3.2: Viewing deployment descriptor information for Web modules (read-only)

To view the deployment descriptor information for a Web module:

1. In the tree on the left side of the console, click **Nodes** -> *hostname* -> **Application Servers** -> *application_server_name* -> **Web Containers** -> **Installed Web modules**to display the Web module view.
2. Click a particular Web module to view its details on right side of the console.
3. Click the link named **View Deployment descriptor (*web*.xml)** where *web* is the application name.

The deployment descriptor information will be displayed.

6.6.8.3.4: Updating Web module configurations with the Web console

During this task, you will update the configuration of anexisting Web module installed on an application server.

To update a Web module configuration:

1. Click **Nodes** -> ***hostname*** -> **Application Servers** -> ***application_server_name*** -> **Web Container** -> **Installed Web Modules** where *application_server_name* is the name of the existing application server.
2. In the list of installed Web modules, click Web module that you want to configure. Its properties will be displayed on the rightside of the console.
3. Modify the properties.
4. When you are finished, click **OK**.
5. Save your configuration.
6. (Optional) To have the configuration take effect:
    1. Stop the server
    2. Start the server again.

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.