JSP files -- table of contents

Development

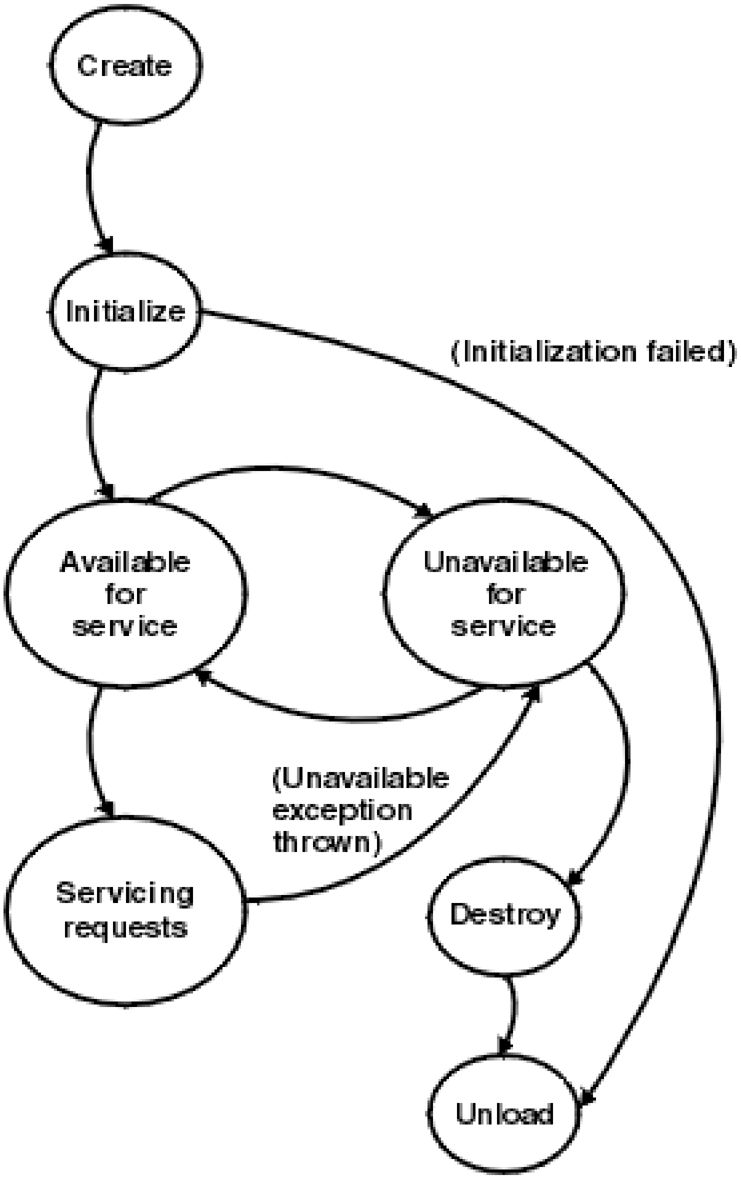
Administration

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.

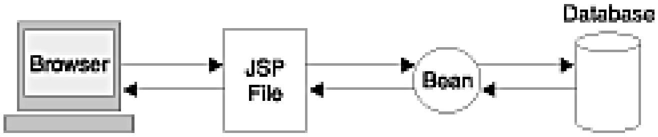
4.2.2.1a: JSP access models

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:

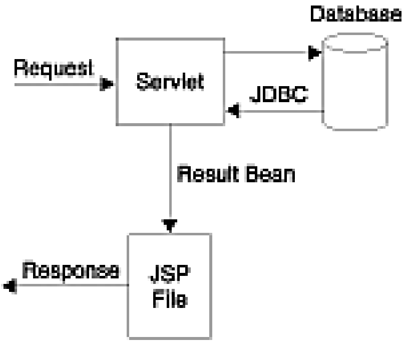
  Request for a JSP file

  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.

It also supports the JSP .91 and JSP 1.0 Specification.Please consult the Related information for the necessary migration of JSP .91 APIs that are deprecated in Version 3.5.

4.2.2.2.1: JSP support for separating logic from presentation

Two interfaces support the JSP technology. These APIs provide a way to separate content generation (business logic) from thepresentation of the content (HTML formatting).

This separation enablesservlets to generate content and store the content (for example, in a bean) inthe request object. The servlet that generated the context generates aresponse by passing the request object to a JSP file that contains the HTMLformatting. The <BEAN> tag provides access to the businesslogic.

The <useBEAN> tag provides access to the businesslogic.

Goal	Interface
Set attributes in the request object.	javax.servlet.http.HttpServletRequest.setAttribute()
Forward a response object to another servlet or JSP file.	javax.servlet.http.RequestDispatcher.forward()

In IBM WebSphere Application Server Version 2.0x, theseinterfaces had different names. You might need to migrate codethat is calling the old interfaces. See the Related informationfor details.

4.2.2.2.2: JSP processors

IBM WebSphere Application Server provides a JSP processor for each supported levelof the JSP specification, .91 and 1.0. Each JSP processor is a **servlet** that you canadd to a Web application to handle all JSP requests pertaining to the Web application.

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.

By specifying either a .91, 1.0 or 1.1 JSP Enabler for each Web application containingJSP files, you configure Web applications to pass JSP files in the Web application folder to the JSP processor correspondingto the JSP specification level of the JSP files.

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

The JSP processor puts the .java, and the .class filein a path specific to theprocessor (see below). The .java and the .class file have the same filename. The processor uses a naming convention that includes adding underscore characters and a suffix to the JSP filename.

For example, if the JSP filename is `simple.jsp`, the generated files are `_simple_xjsp.java` and `_simple_xjsp.class`.

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.

JSP 1.0 processor

Processor servlet name	JSP Servlet
Class name and location	com.sun.jsp.runtime.JspServlet in jsp10.jar
Where processor puts output	*product_installation_root*\temp*servlet_host_name**app_name***???????**

For example, if the JSP file is in:

`c:\WebSphere\AppServer\hosts\default_host\examples\web`

the .java and .class files are put in:

`c:\WebSphere\AppServer\temp\default_host\examples\???????`

JSP .91 processor

Processor servlet name	PageCompileServlet

Class name and location	com.ibm.servlet.jsp.http.pagecompile.PageCompileServlet inibmwebas.jar
Where processor puts output	`product_installation_root\temp\servlet_host_name\app_name\pagecompile` where *product_installation_root* is the path where the Application Server is installed and *app_name* is the name of the application root folder.

For example, if the JSP file is in:

`c:\WebSphere\AppServer\hosts\default_host\examples\web`

the .java and .class files are put in:

`c:\WebSphere\AppServer\temp\default_host\examples\pagecompile`

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.

The process of batch compiling JSP files is different for JSP 0.91 files and JSP 1.0 files. Consult the page corresponding to the JSP level for your files.

4.2.2.2.4.1: Compiling JSP .91 files as a batch

To use the JSP batch compiler for JSP .91 files:

1. Add the following JAR files (found in the Application Server lib directory) to your system classpath:
   - ibmwebas.jar (contains the batch compiler classes)
   - servlet.jar (contains the Java Servlet 2.1 APIs)
2. At an operating system command prompt, enter the following command on a single line:

   ```
   java com.ibm.servlet.jsp.http.pagecompile.jsp.tsx.batch.JspBatch -s sourceRootDir  -t targetRootDir
   -c classPath -l libDirectory -v
   ```

   where:
   - **sourceRootDir**

     The root directory of the paths where the batch JSP compiler will search JSP source files to process. The compiler processes all files with the extension .jsp that are in the source root and its subdirectories.
   - **targetRootDir**

     The root directory of the path where you want the compiler to place the resulting .java and .class files. The non-batch, JSP 0.91 processor (PageCompileServlet) places the .java and .class files in the path:

     `product_installation_root\temp\servlet_host_name\app_name\pagecompile`

     where *product_installation_root* is the path where the Application Server is installed and *app_name* is the name of the application root folder. It is recommended that you specify that path for the target root if you are batch compiling JSPs to run on your production Application Server. However, if you are batch compiling on a different system and plan to move them to the Application Server later, you can specify any valid target root directory.

     If any of the .class files have package names, those names will become the names of subdirectories under the target root. For example, if the .class name is security.login.login.class and the target root is `d:\WebSphere\AppServer\temp\default_host\examples\pagecompile`, the batch compiler places the .java and .class files in `d:\WebSphere\AppServer\temp\default_host\examples\pagecompile\security\login` directory.
   - **classPath**

     An optional parameter that is the fully-qualified path for the classes and Java archives that the compiled classes need. If those resources are in multiple paths, use the semicolon character (;) to separate the path names. You do not need to specify the Application Server JAR files on this parameter.
   - **libDirectory**

     The fully-qualified path to the Application Server ibmwebas.jar (contains the JSP batch compiler and related JSP classes) and servlet.jar (contains the Java Servlet 2.1 APIs). The default path is *product_installation_root*\lib.
   - **-v**

     An optional parameter that causes more trace and progress messages to be displayed.

All of the command parameters, except -v, are required.

Example

Suppose you want to precompile the JSP files associated with the `examples` application, one of the two applications installedwith the application server. If the JSP files are in the path:

```
d:\WebSphere\AppServer\hosts\default_host\examples\web
```

and you want the compiled files to be placed in:

```
d:\WebSphere\AppServer\temp\default_host\examples\pagecompile
```

the command would be (typed on a single line):

```
java com.ibm.servlet.jsp.http.pagecompile.jsp.tsx.batch.JspBatch -s
d:\WebSphere\AppServer\hosts\default_host\examples\web -t
d:\WebSphere\AppServer\temp\default_host\examples\pagecompile -c
d:\WebSphere\AppServer\hosts\default_host\examples\servlets;d:\devcntr\website -l
d:\WebSphere\AppServer\lib
```

4.2.2.2.4.2: Compiling JSP 1.0 files as a batch

To use the JSP batch compiler for JSP 1.0 files, enter the following command on a single line at an operating system command prompt:

```
JspBatchCompiler -adminNodeName <node name> [ -serverName <server name>
                 [-application <application name> [-filename <filename>]]]
                 [-keepgenerated <true|false>]
```

where:

- **adminNodeName**

  This is the name of the node as shown on the Adminstrative Console.
- **serverName**

  [Optional: may only be used if *adminNodeName* is set] This is the name of the Application Server in the WebSphere environment on which you wish to perform this action. Unless you have set up other servers, this will be "Default Server" [Note that from the command-line, you will need to include quote marks around the name of the server if that name comprises two or more words separated by spaces. You do not have to do this if you use the *batchcompiler.config* file described below.]
- **application**

  [Optional: may only be used if *serverName* is set] The name of a particular web application, should you wish to compile only those JSP files under that application.
- **filename**

  [Optional: may only be used if *application* is set] The name of a single file in the web application you selected above, should you wish to compile only a single JSP file in an application.
- **keepgenerated**

  [Optional] If set to "yes" this will keep the generated .java files used for compilation on your server. By default, this is set to "no" and the .java files are all erased after the class files have been compiled.
- **nameServiceHost**

  [Optional] If specified, this parameter and the **nameServicePort** parameter are used in a Model/Clone environment to designate the hostname and port number of the Admin Server to be used in accessing the WebSphere Application Server configuration.
- **nameServicePort**

  [Optional] If specified, this parameter and the **nameServiceHost** parameter are used in a Model/Clone environment to designate the hostname and port number of the Admin Server to be used in accessing the WebSphere Application Server configuration.

In lieu of specifying the parameters in a command line, you may specify them in the *batchcompile.config* file, located in the WebSphere Application Server bin directory. No quotation marks are necessary for any of the variables if you use this file. Any values you enter on the command-line will override the values specified inthe *batchcompile.config* file.

Example

Suppose you want to precompile the JSP files associated with the `examples` application, shipped with WebSphere Application Server. Issue the following command in the appserver bin directory:

```
D:\WebSphere\AppServer\bin>JspBatchCompiler.bat -adminNodeName mynode    -serverName "Default
Server" -application examples
```

You should receive the following response from the server

```
Server name: Default Server
Application Name: examples
   JSP version: 1.0
   docRoot: d:\WebSphere\AppServer\hosts\default_host\examples\web
   Application Classpath: d:\WebSphere\AppServer\hosts\default_host\examples\servlets;
   Application output dir: d:\WebSphere\AppServer/temp/default_host/examples
   URL: .jsp
   URL: .jsv
   URL: .jsw
Attempting to compile: d:\WebSphere\AppServer\hosts\default_host\examples\web\debug_error.jsp
Compilation successful
Attempting to compile: d:\WebSphere\AppServer\hosts\default_host\examples\web\HelloHTML.jsp
Compilation successful
    .       .       .Attempting to compile:
d:\WebSphere\AppServer\hosts\default_host\examples\web\StockQuoteWMLRequest.jspCompilation
successful
Attempting to compile:
d:\WebSphere\AppServer\hosts\default_host\examples\web\StockQuoteWMLResponse.jspCompilation
```

```
    successful
```

If you look in the appserver temp directory, you should see a directory named examples. All of the compiled class files for the examples application will be in this directory.

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 ℹ 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 (JSP .91 or JSP 1.0)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.
<BEAN> tag	Use the <BEAN> 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 .91) or (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.

<SERVLET> tags

Using the <SERVLET> tag is one method for embedding a servletwithin a JSP file.

NCSA tags

You might have legacy SHTML files that contain NCSA tags for server-side includes. If the IBM WebSphere Application Server Version 3.5 supports the NCSAtags in your SHTML files, you can convert the SHTML files to JSP files andretain the NCSA tags.

4.2.2.3.1: JSP syntax: JSP 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

For more information on the JSP 1.1 technologies, view the Tomcatdocumentation at the Sun™ site.

The general syntax of the JSP directive is:

`<%@ directive_name ="value" %>`

where the valid directive names are:

- **language**

  The scripting language used in the file. At this time, the onlyvalid value and the default value is `java` (the Java programminglanguage). The scope of this directive is the JSP file.When used more than once, only the first occurrence of the directive issignificant. An example:

  ```
  <%@ language ="java" %>
  ```

- **method**

  The name of the method generated by the embedded Java code(scriptlet). The generated code becomes the body of the specifiedmethod name. The default method is `service`. When usedmore than once, only the first occurrence of the directive issignificant. An example:

  ```
  <%@ method ="doPost" %>
  ```

- **import**

  A comma-separated list of Java language package names or class names thatthe servlet imports. This directive can be specified multiple timeswithin a JSP file to import different packages. An example:

  ```
  <%@ import ="java.io.*,java.util.Hashtable" %>
  ```

- **content_type**

  The MIME type of the generated response. The default value is`text/html`. This information is used to generate the response header. When used more than once, only the first occurrence of thisdirective is significant. This directive can be used to specify the character set in which the pageis to be encoded. An example:

  ```
  <%@ content_type ="text/html; charset=iso-8859-1" %>
  ```

- **implements**

  A comma-separated list of Java language interfaces that the generatedservlet implements. You can use this directive more than once within aJSP file to implement different interfaces. An example:

  ```
  <%@ implements ="javax.servlet.http.HttpSessionContext" %>
  ```

- **extends**

  The name of the Java language class that the servlet extends. Theclass must be a valid class and does not have to be a servlet class.The scope of this directive is the JSP file. When used morethan once, only the first occurrence of the directive is significant.An example:

```
<%@ extends ="javax.servlet.http.HttpServlet" %>
```

4.2.2.3.2: JSP syntax: Class-wide variables and methods

Use the <SCRIPT> and </SCRIPT> tags to declareclass-wide variables and class-wide methods for the servlet class. Thegeneral syntax is:

```
<script runat=server>// code for class-wide variables and methods</script>
```

The attribute `runat=server` is required and indicates that thetag is for server-side processing. An example of specifying class-widevariables and methods:

```
<script runat=server>// class-wide variables    init i = 0;     String foo = "Hello";// class-wide
methods      private void foo() {// code for the method}  </script>
```

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.

The scriptlet can use a set of predefined variables thatcorrespond to essential servlet, output, and input classes:

- **request**

  The servlet request class defined byjavax.servlet.http.HttpServletRequest
- **response**

  The servlet response class defined byjavax.servlet.http.HttpServletResponse
- **out**

  The output writer class defined by java.io.PrintWriter. The content written to the writer is the client response.
- **in**

  The input reader class defined by java.io.BufferedReader

An example:

```
<%foo = request.getParameter("Name");out.println(foo);%>
```

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.

4.2.2.3.5.1a: JSP .91 syntax: <BEAN> tag syntax

```
<bean name="bean_name" varname="local_bean_name"   type ="class_or_interface_name"
introspect="yes|no"    beanName="ser_filename" create="yes|no"   scope="request|session|userprofile"
></bean>
```

where the attributes are:

- **name**

  The name used to look up the bean in the appropriate scope (specified bythe scope attribute). For example, this might be the session key valuewith which the bean is stored. The value is case-sensitive.
- **varname**

  The name used elsewhere within the JSP file to refer to the bean.This attribute is optional. The default value is the value of the nameattribute. The value is case-sensitive.
- **type**

  The name of the bean class file. This name is used todeclare the bean instance in the code. The default value is the typeObject. The value is case-sensitive.
- **introspect**

  When the value is `yes`, the JSP processor examines all requestproperties and calls the set property methods (passed in the BeanInfo) thatmatch the request properties. The default value of this attribute is`yes`.
- **beanName**

  The name of the bean class file, the bean package name, or theserialized file (.ser file) that contains the bean. (This nameis given to the bean instantiator.) This attribute is used only whenthe bean is not present in the specified scope and the create attribute is setto `yes`. The value is case-sensitive.

  The path of the file must be specified in the Web application classpath.
- **create**

  When the value is `yes`, the JSP processor creates an instance ofthe bean if the processor does not find the bean within the specifiedscope. The default value is `yes`.
- **scope**

  The lifetime of the bean. This attribute is optional and thedefault value is `request`. The valid values are:
  - request - The bean is added to the request object by a servlet thatinvokes the JSP file using the APIs described in JSP API.If the bean is not part of the request context, the bean is created and stored in the request contextunless the create attribute is set to `no`.
  - session - If the bean is present in the current session, the bean isreused. If the bean is not present, it is created and stored as part ofthe session if the create attribute is set to `yes`.
  - userprofile - This attribute value is an IBM extension to JSP 0.91 and causes the user profile to be retrieved from the servlet requestobject, cast to the specified type, and introspected. If a type is notspecified, the default type is

    `com.ibm.websphere.UserProfile`

    .The create attribute is ignored.

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

4.2.2.3.5.2a: JSP .91 syntax: Accessing bean properties

After specifying the <BEAN> tag, you can access the bean at any pointwithin the JSP file. There are three methods for accessing beanproperties:

- Using a JSP scriptlet
- Using a JSP expression
- Using the <INSERT> tag (as described in the JSP .91 tags for variable data)

An example:

```
<!-- The bean declaration --> <bean name="foobar" type="FooClass" scope="request" >    <param
name="fooProperty" value="fooValue"></bean> <!-- Later in the file, some HTML content that includes
JSP syntax that calls a method of the bean --> <p>The name of the row is <%= foobar.getRowName()
%>.</p>
```

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.

4.2.2.3.5.3a: JSP .91 syntax: Setting bean properties

You can set the bean properties by using the <PARAM> tag within the <BEAN> tag. The <PARAM> tag specifies a list of properties and the corresponding values. The properties areautomatically set in the bean using introspection. The properties areset once when the bean is instantiated. The <PARAM> tag syntax is:

`<PARAM name="`*`property_name`*`" value="`*`property_value`*`">`

This syntax is an IBM extension to the JSP 0.91 <PARAM> tag. The IBM syntax is consistent with the syntax of the <PARAM> tag used within the <SERVLET> and <APPLET> tags.

In addition to using the <param> tag to set bean properties,there are three other methods:

- Specifying query parameters when requesting the URL of the JSPfile that contains the bean. The introspect attribute must be set toyes. An example:

  ```
  http://www.myserver.com/signon.jsp?name=jones&password=d13x
  ```

  where the bean property name will be set to `jones`.
- Specifying the properties as parameters submitted through an HTML<FORM> tag. The JSP method directive must be set to`post`. The action attribute is set to the URL of the JSP filethat invokes the bean. The introspect attribute must be set toyes. An example:

  ```
  <form action="http://www.myserver.com/SearchSite.jsp" method="post">  <input type="text"
  name="Search for: ">  <input type="submit"></form>
  ```
- Using JSP syntax to set the bean property

4.2.2.3.5a: JSP .91 syntax: BEAN tags

Use the <BEAN> tag to create an instance of a bean that will beaccessed elsewhere within the JSP file. Then use JSP tags for variable data (such as the <INSERT> tag described later in this document) to access the bean.

The 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.

4.2.2.3.6: Supported NCSA tag reference

The product supports the following NCSA tags through their use in JSP files:

- config
- echo var=*variable* (see below)
- exec
- filesize
- include
- lastmodified
- Commands for formatting size and date outputs

For the echo command, the product supports thestandard server-side include (SSI) environment variables and Common GatewayInterface (CGI) environment variables.

The SSI environment variables

Variable	Description
DATE_GMT	The current date and local time zone in Greenwich mean time (GMT)
DATE_LOCAL	The current date and local time zone
DOCUMENT_NAME	The current filename
DOCUMENT_URI	The path to the document (such as, /docs/tutorials/index.shtml)
QUERY_STRING_UNESCAPED	The unescaped version of any search query the client sent, with all shellspecial characters escaped with the \ character
LAST_MODIFIED	The last date the current document was changed

CGI environment variables

Variable	Description
AUTH_TYPE	The protocol-specific authentication method used to validate the user, ifthe server supports user authentication and the script is protected
CONTENT_LENGTH	The length of the content, as specified by the remote host
CONTENT_TYPE	The data content type for queries that have information attached(such as HTTP POST and PUT)
GATEWAY_INTERFACE	The revision level of the CGI specification to which the server complies
PATH_INFO	The extra path information given by the client in this request. Theextra information follows the virtual pathname of the CGI script.
PATH_TRANSLATED	The server provides a translated version of PATH_INFO, which takes the pathand performs any virtual-to-physical mapping.
QUERY_STRING	The information that follows the ? symbol in the URL request for a script
REMOTE_HOST	The hostname of the remote host sending the request. If the serverdoes not have this information, the server should set REMOTE_ADDR and leaveREMOTE_HOST unset.
REMOTE_ADDR	The IP address of the remote host sending the request
REMOTE_IDENT	If the HTTP server supports RFC 931 identification, the remote usernameretrieved from the server

REMOTE_USER	The username used for authentication, if the server supports userauthentication and the script is protected
REQUEST_METHOD	The method with which this request was made. Methods include HTTP, GET, HEAD,POST, and so on
SCRIPT_NAME	The virtual path to the script being run. This variable is used forself-referencing URLs
SERVER_NAME	The IP address, hostname, or Domain Name Server (DNS) alias of the server
SERVER_PORT	The port number to which the request was sent
SERVER_PROTOCOL	The name and revision level of the protocol used to format this request
SERVER_SOFTWARE	The name and version of the server answering the request

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"
```

> ℹ 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>
```

4.2.2.3.8: IBM extensions to JSP .91 syntax

Refer to the Sun JSP .91 specification for the base JavaServer Pages (JSP)APIs. IBM WebSphere Application Server provides several extensions to the baseAPIs.

For JSP .91, the extensions belong to these categories:

Extension	Use
Syntax for variable data	Put variable fields in JSP files and have servlets and JavaBeans dynamically replace the variables with values from a database when the JSP output is returned to the browser.
Syntax for database access	Add a database connection to 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 the user at request time, or can be hardcoded within the JSP file.

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

See the Related information for syntax details.

4.2.2.3.8.1: JSP .91 syntax: Tags for variable data

The variable data syntax enables you to put variable fields on your HTML page and have your servlets and JavaBeans dynamically 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
Embed variables in a JSP file	<INSERT>	This is the base tag for specifyingvariable fields.
Repeating a block of HTML tagging that contains the <INSERT> tags and the HTML tags for formatting content	<REPEAT>	Use the <REPEAT> tag to iterate over a database query results set. The <REPEAT> tag iterates from the start value to the end value until one of the following conditions is met: ● The end value is reached. ● An ArrayIndexOutofBoundsException is thrown. The output of a <REPEAT> block is buffered until the block completes. If an exception is thrown before a block completes, no output is written for that block.

The above tags are designed to pass entact through HTML authoringtools. Each tag has a corresponding end tag. Each tag iscase-insensitive, but some of the tag attributes are case-sensitive.

4.2.2.3.8.1.1: JSP .91 syntax: <INSERT> tag syntax

```
<insert requestparm=pvalue requestattr=avalue bean=name
property=property_name(optional_index).subproperty_name(optional_index)
default=value_when_null></insert>
```

where:

- **requestparm**

  The parameter to access within the request object. This attributeis case-sensitive and cannot be used with the bean and propertyattributes.

- **requestattr**

  The attribute to access within the request object. The attributewould have been set using the setAttribute method. This attribute iscase-sensitive and cannot be used with the bean and propertyattributes.

- **bean**

  The name of the JavaBean declared by a <BEAN> tag within the JSPfile. The value of this attribute is case-sensitive.

  When the bean attribute is specified but the property attribute is notspecified, the entire bean is used in the substitution. For example, ifthe bean is type String and the property is not specified, the value of thestring is substituted.

- **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. This attribute cannot be used with the requestparm andrequestattr attributes.

- **default**

  An optional string to display when the value of the bean property isnull. If the string contains more than one word, the string must beenclosed within a pair of double quotes (such as "HelpDesk number").The value of this attribute is case-sensitive. If a value is notspecified, an empty string is substituted when the value of the property isnull.

Use the alternate syntax instead if you need to embed the INSERT tag withinanother HTML tag.

4.2.2.3.8.1.1a: JSP .91 syntax: Alternate syntax for the <INSERT> tag

The HTML standard does not permit embedding HTML tags within HTML tags. Consequently, you cannot embed the <INSERT> tag within another HTML tag, for example, the anchor tag (<A>). Instead, use the alternate syntax.

To use the alternate syntax:

1. Use the <INSERT> and </INSERT> to enclose the HTML tag inwhich substitution is to take place.
2. Specify the bean and property attributes:
   - To specify the bean and property attributes, use the form:

     `$(bean=b property=p default=d)`

     where *b*, *p*, and *d* are values as described forthe <INSERT> tag.
   - To specify the requestparm attribute, use the form

     `$(requestparm=r default=d)`

     where *r* and *d* are values as described for the <INSERT> tag.
   - To specify the requestattr attribute, use the form

     `$(requestattr=r default=d)`

     where *r* and *d* are values as described for the <INSERT> tag.

4.2.2.3.8.1.1b: Example: JSP .91 syntax: INSERT tag syntax

Regular syntax

```
<insert bean=userProfile property=username></insert><insert requestparm=company default="IBM
Corporation"></insert><insert requestattr=ceo default="Company CEO"></insert><insert
bean=userProfile property=lastconnectiondate.month></insert>
```

In most cases, the value of the property attribute will be just theproperty name. However, you access a property of a property(sub-property) by specifying 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 <REPEAT> tag. Some examples of using the full form of the property attribute:

```
<insert bean=staffQuery property=address(currentAddressIndex)></insert><insert bean=shoppingCart
property=items(4).price></insert><insert bean=fooBean property=foo(2).bat(3).boo.far></insert>
```

Alternate syntax

```
<insert>  <img src=$(bean=productAds property=sale default=default.gif)></insert> <insert>  <a
href="http://www.myserver.com/map/showmap.cgi?country=$(requestparm=country
default=usa)&city$(requestparm=city default="Research Triangle Park")  &email=$(bean=userInfo
property=email)>Show map of city</a></insert>
```

4.2.2.3.8.1.2: JSP .91 syntax: <REPEAT> tag syntax

`<repeat index=name start=starting_index end=ending_index></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.

4.2.2.3.8.1.2a: JSP .91 syntax: <REPEAT> tag results set and the associated bean

The <REPEAT> tag 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<DBQUERY> tag). 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 <DBQUERY> tag 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 <REPEAT> tag iterates over the rows (beginning at the start row).

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

Static Bean Example	**<DBQUERY> Bean Example**

myBean.class
`// Code to get a connection// Code to get the data   Select * from myTable;// Code to close the connection`
JSP file
`<repeat index=abc>  <insert bean="myBean"    property="col1(abc)">  </insert></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 <REPEAT> tag is used. You can also omit the index on the <REPEAT> tag.
● The <REPEAT> tag iterates over the bean properties row by row, beginning with the start row.
	JSP file
	`<dbconnect id="conn"userid="alice"passwd="test"url="jdbc:db2:sample"driver="COM.ibm.db2.jdbc.app.DB2Driver"<`
	`id="dynamic" connection="conn" >  Select * from myTable;</dbquery><repeat index=abc>  <insert bean="dynamic"`
	`property="col1(abc)">  </insert></repeat>`
	● The bean (dynamic) is generated by the <DBQUERY> tag and does not exist until the tag 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 <REPEAT> tag is used. You can also omit the index on the <REPEAT> tag.
	● The <REPEAT> tag 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 <REPEAT> tag. 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><repeat>  <tr><td><insert bean=serviceLocationsQuery property=city></insert></tr></td>  <tr><td><insert
bean=serviceLocationsQuery property=address></insert></tr></td>  <tr><td><insert bean=serviceLocationsQuery
property=telephone></insert></tr></td></repeat></table>
```

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

```
<table><repeat index=myIndex start=0 end=2147483647>  <tr><td><insert bean=serviceLocationsQuery
property=city(myIndex)></insert></tr></td>  <tr><td><insert bean=serviceLocationsQuery
property=address(myIndex)></insert></tr></td>  <tr><td><insert bean=serviceLocationsQuery
property=telephone(myIndex)></insert></tr></td></repeat></table>
```

The JSP compiler for the Application Server Version 3 is designed to prevent the ArrayIndexOutofBoundsException with explicit indexing. Consequently, you do not need to place JSP variable data syntax before the <INSERT> tag to check the validity of the index.

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><repeat index=myIndex end=299>  <tr><td><insert bean=serviceLocationsQuery property=city></insert></tr></td>
<tr><td><insert bean=serviceLocationsQuery property=address(myIndex)></insert></tr></td>  <tr><td><insert
bean=serviceLocationsQuery property=telephone(myIndex)></insert></tr></td></repeat></table>
```

Nesting <REPEAT> tags

You can nest <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<REPEAT> blocks are nested to display the list of songs on each compactdisc in the user's shopping cart.

```
<repeat index=cdindex>  <h1><insert bean=shoppingCart property=cds.title></insert></h1>  <table>  <repeat>    <tr><td><insert
bean=shoppingCart property=cds(cdindex).playlist></insert>    </td></tr>  </table>  </repeat></repeat>
```

4.2.2.3.8.2: JSP .91 syntax: JSP tags for database access

The Application Server Version 3.5 extends JSP 0.91 support by providing a set of tags for database access. These HTML-like tags make it simple to add a database connection to 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 the user at request time or hardcoded 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	<DBCONNECT>	The <DBCONNECT> tag does not establish the connection. Instead, the <DBQUERY> and <DBMODIFY> tags are used to reference a <DBCONNECT> tag in the same JSP file and establish the connection. When the JSP file is compiled into a servlet, the Java processor adds the Java coding for the <DBCONNECT> tag to the servlet's service() method, which means a new database connection is created for each request for the JSP file. **Examples:** Employee.jsp example
Avoid hard coding the user ID and password in the <DBCONNECT> tag	<USERID> and <PASSWD>	Use the <USERID> and <PASSWD> tags to accept user input for the values and then add that data to the request object where it can be accessed by a JSP file (such as the Employee.jsp example) that requests the database connection. The <USERID> and <PASSWD> tags must be used within a <DBCONNECT> tag. **Examples:** None
Establish a connection to a database, submit database queries, and return the results set.	<DBQUERY>	The <DBQUERY> tag: 1. References a <DBCONNECT> tag in the same JSP file and uses the information provided by that tag to determine the database URL and driver. The user ID and password are also obtained from the <DBCONNECT> tag if those values are provided in the <DBCONNECT> tag. 2. Establishes a new connection 3. Retrieves and caches data in the results object 4. Closes the connection (releases the

		connection resource) **Examples:** Basic example Employee.jsp EmployeeRepeatResults.jsp
Establish a connection to a database and then add records to a database table.	<DBMODIFY>	The <DBMODIFY> tag: 1. References a <DBCONNECT> tag in the same JSP file and uses the information provided by that tag to determine the database URL and driver. The user ID and password are also obtained from the <DBCONNECT> tag if those values are provided in the <DBCONNECT> tag. 2. Establishes a new connection 3. Updates a table in the database 4. Closes the connection (releases the connection resource) **Examples:** Basic example EmployeeRepeatResults.jsp
Display query results	<REPEAT> and <INSERT> tags	The <REPEAT> tag loops through each of the rows in the query results. The <INSERT> tag uses the query results object (for the <DBQUERY> tag whose identifier is specified by the <INSERT> bean attribute) and the appropriate column name (specified by the <INSERT> property attribute) to retrieve the value. **Examples:** Basic example

4.2.2.3.8.2.1: JSP .91 syntax: <DBCONNECT> tag syntax

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

where:

- **id**

  A required identifier for this tag. The scope is the JSP file. This identifier is referenced by the connection attribute of the <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 <USERID> and <PASSWD> for an alternative to hardcoding this information in the JSP file.
- **passwd**

  An optional attribute that specifies the user password for the userid. (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 <USERID> and <PASSWD> for an alternative to hardcoding this attribute in the JSP file.
- **url** and **driver**

  To establish a database connection, the URL and driver must be provided. If these attributes are not specified in the <DBCONNECT> tag, the xmlref attribute or the jndiname attribute must be specified.

  The Application Server Version 3 supports connection to JDBC databases and ODBC databases. When connecting to an ODBC database, you can use the Sun JDBC-to-ODBC bridge driver included in the Java Development Kit (JDK) or another 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.

  For a connection to a JDBC database, the URL consists of the following colon-separated elements: jdbc, the sub-protocol name, and the name of the database table 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"
  ```

  If the database is an ODBC database, you can use an ODBC driver or the 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"
  ```

  > ℹ 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.

  > ℹ If your JSP accesses a different JDBC or ODBC database than the one the Application Server uses for its repository, ensure that you add the JDBC or ODBC driver for the other database to the Application Server's classpath.
- **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 <DBCONNECT> tag or in the file specified by the xmlref tag.
- **xmlref**

  A file (in XML format) that contains the URL, driver, user ID, password information needed for a connection. This mechanism provides Web administrators an alternative method for specifying the user ID and password. It is an alternative to hardcoding the information in a <DBCONNECT> tag or reading the information from the request object parameters. This is useful when third-party vendors develop your JSP files and when you need to make quick changes or test an application with a different data source.

  When the JSP compiler processes the <DBCONNECT> tag, it reads all of the specified tag attributes. If any of the required attributes are missing, the compiler checks for an xmlref attribute. If the attribute is specified, the compiler reads the configuration file.

  The xmlref takes precedence over the <DBCONNECT> tag. For example, if the <DBCONNECT> tag and the xmlref file include values for the URL and the the driver, the values in the xmlref file are used.

The configuration file can have any filename and extension that is valid for the operating system. Place the file in the same directory as the JSP that contains the referring <DBCONNECT> tag. An example of a configuration file is:

```
<?xml version="1.0" ?><db-info>  <url>jdbc:odbc:autos</url>  <user-id>smith</user-id>
<dbDriver>sun.jdbc.odbc.JdbcOdbcDriver</dbDriver>  <password>v598m</password>
<jndiName>jdbc/demo/sample</jndiName></db-info>
```

All of the elements shown in the example XML file need to be specified. However, an empty element (such as <url></url>) is valid.

4.2.2.3.8.2.2: JSP .91 syntax: <USERID> and <PASSWD> tag syntax

```
<dbconnect id="connection_id"   <font color="red"><userid></font><insert
requestparm="userid"></insert><font color="red"></userid></font>       <font
color="red"><passwd></font><insert requestparm="passwd"></insert><font color="red"></passwd></font>
url="protocol:database_name:database_table"     driver="JDBC_driver_name"> </dbconnect>
```

where:

- **<INSERT>**

  This tag is a JSP tag for including variable data. See JSP tags for variable data.
- **userid** tag

  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.

  See the Login.jsp and the Employee.jsp examples for an illustration of how to set the USERID and PASSWD using parameters in the request object. The request parameters are set using an HTML form (Login.jsp). In the Employee.jsp, the values of the parameters are passed as hidden form values to the EmployeeRepeatResults.jsp.
- **passwd** tag

  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.

4.2.2.3.8.2.3: JSP .91 syntax: <DBQUERY> tag

```
<!-- SELECT commands and (optional) JSP syntax can be placed within the DBQUERY tag. --><!-- Any
other syntax, including HTML comments, are not valid. --><dbquery id="query_id"
connection="connection_id" limit="value" ></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 <INSERT> tag 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 <DBCONNECT> tag in this JSP file. That <DBCONNECT> tag 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 and the effective limit is determined by the number of records and the system caching capability.
- SELECT command and JSP syntax

  Because the <DBQUERY> tag must return a results set, the only valid SQL command is SELECT. 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.

4.2.2.3.8.2.3a: Example: JSP .91 syntax: <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 <INSERT> tag to embed a variable data field. The value of that field is based on user input.

```
<dbquery id="empqs" connection="conn" >select * from Employee where WORKDEPT='<INSERT
requestparm="WORKDEPT"></INSERT>'</dbquery>
```

4.2.2.3.8.2.4: JSP .91 syntax: <DBMODIFY> tag syntax

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

where:

- **connection**

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

  Refer to your database documentation for valid database commands.

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 <INSERT> tags.

```
<dbmodify connection="conn" >insert into EMPLOYEE
(EMPNO,FIRSTNME,MIDINIT,LASTNAME,WORKDEPT,EDLEVEL)values        ('<INSERT
requestparm="EMPNO"></INSERT>',       '<INSERT requestparm="FIRSTNME"></INSERT>',     '<INSERT
requestparm="MIDINIT"></INSERT>',      '<INSERT requestparm="LASTNAME"></INSERT>',     '<INSERT
requestparm="WORKDEPT"></INSERT>',     <INSERT requestparm="EDLEVEL"></INSERT>)</dbmodify>
```

The EmployeeRepeatResults.jsp example illustrates this tag.

Displaying query results

To display the query results, use the <REPEAT> and <INSERT> tags. The <REPEAT> tag loops through each of the rows in the query results. The <INSERT> tag uses the query results object (for the <DBQUERY> tag whose identifier is specified by the <INSERT> bean attribute) and the appropriate column name (specified by the <INSERT> property attribute) to retrieve the value. An example is:

```
<repeat><tr>    <td><INSERT bean="empqs" property="EMPNO"></INSERT>     <INSERT bean="empqs"
property="FIRSTNME"></INSERT>      <INSERT bean="empqs" property="WORKDEPT"></INSERT>      <INSERT
bean="empqs" property="EDLEVEL"></INSERT>       </td></tr></repeat>
```

JSP 0.91 APIs and migration

Two interfaces support the JSP 0.91 technology. TheseAPIs provide a way to separate content generation (business logic) from thepresentation of the content (HTML formatting). This separation enablesservlets to generate content and store the content (for example, in a bean) inthe request object. The servlet that generated the context generates aresponse by passing the request object to a JSP file that contains the HTMLformatting. The <BEAN> tag provides access to the businesslogic.

The interfaces that supported JSP 0.91 for the Application Server Version 3 are:

- javax.servlet.http.HttpServletRequest.setAttribute()

  Supports setting attributes in the request object. For the Application Server Version 2, this interface was com.sun.server.http.HttpServiceRequest.setAttribute().
- javax.servlet.http.RequestDispatcher.forward()

  Supports forwarding a response object to another servlet or JSP. For the Application Server Version 2, this interface was com.sun.server.http.HttpServiceResponse.callPage().

4.2.2.3.8.2.4a: Example: JSP .91 syntax: <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 <INSERT> tags.

```
<dbmodify connection="conn" >insert into EMPLOYEE
(EMPNO,FIRSTNME,MIDINIT,LASTNAME,WORKDEPT,EDLEVEL)values        ('<INSERT
requestparm="EMPNO"></INSERT>',       '<INSERT requestparm="FIRSTNME"></INSERT>',     '<INSERT
requestparm="MIDINIT"></INSERT>',      '<INSERT requestparm="LASTNAME"></INSERT>',     '<INSERT
requestparm="WORKDEPT"></INSERT>',     <INSERT requestparm="EDLEVEL"></INSERT>)</dbmodify>
```

4.2.2.3.8.2.5a: Example: JSP .91 syntax: <INSERT> and <REPEAT> tags

```
<repeat><tr>    <td><INSERT bean="empqs" property="EMPNO"></INSERT>     <INSERT bean="empqs"
property="FIRSTNME"></INSERT>      <INSERT bean="empqs" property="WORKDEPT"></INSERT>      <INSERT
bean="empqs" property="EDLEVEL"></INSERT>       </td></tr></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="employee.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><% String userID    = request.getParameter("USERID"); %><% String
passWord  = request.getParameter("PASSWD"); %><%-- 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"><tsx:userid><%=userID%></tsx:userid><tsx:passwd><%=passWord%></tsx: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="<%=userID%>"><INPUT
TYPE="hidden" NAME="PASSWD" VALUE="<%=passWord%>"></FORM><HR><FORM NAME="EmployeeForm"
ACTION="employeeRepeatResults.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="<%=userID%>"><INPUT TYPE="hidden" NAME="PASSWD"
VALUE="<%=passWord%>"></FORM><HR></BODY></HTML>
```

JSP code example - EmployeeRepeatResults

```
<HTML><HEAD><TITLE>JSP Employee Results</TITLE></HEAD><H1><CENTER>EMPLOYEE RESULTS</CENTER></H1><BODY><% String userID      =
request.getParameter("USERID"); %><% String passWord    = request.getParameter("PASSWD"); %><% String empno       =
request.getParameter("EMPNO"); %><% String firstnme    = request.getParameter("FIRSTNME"); %><% String midinit     =
request.getParameter("MIDINIT"); %><% String lastname    = request.getParameter("LASTNAME"); %><% String workdept    =
request.getParameter("WORKDEPT"); %><% String edlevel     = request.getParameter("EDLEVEL"); %><!-- Get a connection to the local
DB2 database using parameters from login.jsp --><tsx:dbconnect id="conn" url="jdbc:db2:sample"
driver="COM.ibm.db2.jdbc.app.DB2Driver"><tsx:userid><%=userID%></tsx:userid><tsx:passwd><%=passWord%></tsx:passwd></tsx:dbconnect><%
if  ( ( request.getParameter("Submit")).equals("Update") ) { %><tsx:dbmodify connection="conn" >   INSERT INTO  EMPLOYEE
(EMPNO,FIRSTNME,MIDINIT,LASTNAME,WORKDEPT,EDLEVEL)   VALUES    ( '<%=empno%>',    '<%=firstnme%>',     '<%=midinit%>',
'<%=lastname%>',    '<%=workdept%>',    <%=edlevel%>)    </tsx:dbmodify> <B><UL>UPDATE SUCCESSFUL</UL></B> <BR><BR><tsx:dbquery
id="qs" connection="conn" >   select * from Employee  where  WORKDEPT= '<%=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= '<%=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>
```

4.2.2.3b: JSP .91 examples

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

(Login.jsp)	An interface for logging in to the application
(Employee.jsp)	A dialog for querying and updating database records
(EmployeeRepeatResults.jsp)	A dialog for displaying update confirmations and query results

6.6.7: Administering servlet engines (overview)

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

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

6.6.7.0: Servlet engine properties

Application Server

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

Current State

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

Desired state

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

Max Connections

Specifies the maximum number of concurrent resource requests to allow.

Max Connections in use

Specifies the Max Connections value currently in use.

Port

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

Port in use

Specifies Port value currently in use.

Queue Type (Transport Type)

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

OSE	For routing requests locally. It is also for using remote OSE for Advanced Edition.
HTTP	Not recommended at this time
None	For use with thin servlet redirector for Advanced Edition

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

Clone Index

Specifies a unique numerical identifier for this servlet engine instance.If there are multiple clones of a servlet engine, each clone instance will havea unique clone index value.

Native Log File

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

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

Queue Name

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

Select Log File Mask

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

Transport Type

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

- Local pipes

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

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

Queue Type in use

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

Servlet Engine Mode FP 3.5.2

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

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

For example, if the path for a servlet is:

```
default_host/WebSphereSamples/servlet
```

then change it to:

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

Servlet Engine Name

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

Start Time

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

6.6.7.1: Administering servlet engines with the Java administrative console

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

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

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

6.6.7.1.1: Configuring new servlet engines with the Java administrative console

The product offers several ways to configure new servlet engines:

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

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

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

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

6.6.7.3: Administering servlet engines with the Web console

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

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

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

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

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

6.6.7.4: Property files pertaining to servlet engines

The servlet engine properties are in file:

- *servlet_engine.properties*

This file is located in directory:

`<WebSphere/Appserver>/properties`

6.6.8: Administering Web applications (overview)

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

Approaches to configuring Web applications

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

- From the bottom up
- From the top down

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

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

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

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

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

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

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

Web applications inside enterprise applications

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

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

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

Configuring Web applications directly in WebSphere systems administration

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

- Classpath

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

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

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

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

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

The administrator can also specify properties such as:

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

Classpath considerations

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

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

Identifying a welcome page for the Web application

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

```
/webapp/myapp
```

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

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

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

FP 3.5.2 Converting WAR files

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

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

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

Utilizing servlets available from WebSphere

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

6.6.8.0: Web application properties

Attributes

Specifies servlet context attributes for the entire Web application.

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

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

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

Auto Reload

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

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

Classpath

Specifies the classpath for the Web application.

Classpath in use

Specifies the classpath currently in use for the Web application.

Current State

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

Desired State

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

Description

Specifies a description of the Web application.

Document Root

Specifies the document root of the Web application.

Enabled

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

Error Page FP 3.5.2 (changed)

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

Consists of:

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

Example values:

- Status Code: 404

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

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

Full Web Path in use

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

MIME Table FP 3.5.2

Specifies mappings between extensions and MIME types. Consists of:

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

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

Reload Interval

Specifies the interval between reloads of the web application.

Specify the value in *seconds*

6.6.8.1: Administering Web applications with the Java administrative console

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

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

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

6.6.8.1.1: Configuring new Web applications

The product offers several ways to configure new Web applications:

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

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

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

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

      For example, if you would like users to type

      `http://`*`default_host_alias`*`/webapp/mywebapp`

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

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

FP
3.5.2

6.6.8.1.6: Converting WAR files with the Java administrative console

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

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

You will need to specify the following information:

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

6.6.8.3: Administering Web applications with the Web console

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

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

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

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

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

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.

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.