IBM VisualAge® for Java™, Version 3.5

JSP/Servlet Development Environment

Note!
Before using this information and the product it supports, be sure to read the general information under **Notices**.

Edition notice

This edition applies to Version 3.5 of IBM VisualAge for Java and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 1. JSP and servlets

A servlet is a Java program that plugs into a Web server. A Web server can be extended to host servlets through a servlet engine. Servlets make it easy to expand from client/single-server applications to multi-tier applications. Servlets allow businesses to connect databases to the Web.

Servlets greatly improve portability. Because servlets are written in Java, they are portable across platforms; they do not have to be recompiled for different operating systems. The servlet interface is a standard, so servlets can be moved from one servlet engine to another, as long as the servlets do not use vendor extensions. However, even if vendor extensions are used, the servlet engine will support a variety of Web servers, which means that the servlet will not be locked into a single platform. Consequently, programmers can develop on an operating system that has good tool support, such as Windows NT, and then deploy on an operating system with good scalability, such as AIX.

You can develop, debug, and deploy servlets within the VisualAge for Java Integrated Development Environment (IDE). In the IDE, you can set breakpoints within servlet objects, and step through code to make changes that are dynamically folded into the running servlet on a running server, without having to restart each time.

Although a servlet can be a completely self-contained program, the task of generating dynamic content should be split into the following two parts, to ease server-side programming:

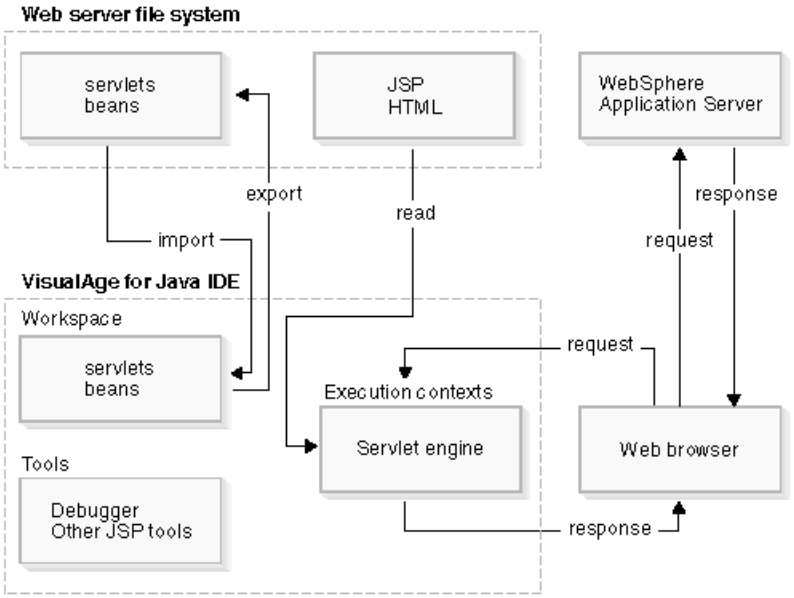
- The business logic (content generation), which governs the relationship between input, processing, and output
- The presentation logic (content presentation, or graphic design rules), which determines how information is presented to the user

In this scenario, business logic can be handled by Java beans, and presentation logic can be handled by JavaServer Pages, while the servlet handles the HTTP protocol. With JavaServer Pages technology, you can efficiently separate the business logic of an application from its presentation logic.

VisualAge for Java provides the WebSphere Test Environment (WTE), which contains the WebSphere Application Server Advanced Edition run-time environment. This unit test environment allows you to efficiently develop your servlets and JSP files for WebSphere. **Important!** The WTE encompasses the JSP file, servlet, and EJB run times and unit testing environment; it does *not* contain the entire WebSphere Advanced run time.

The WebSphere Application Server is IBM's Java servlet-based Web application server that helps you deploy and manage Web applications. WebSphere Application Server is a Web server plug-in based on a server-side Java programming model that uses servlets, EJB beans, and JavaServer Pages technology.

VisualAge for Java also provides the Create Servlet SmartGuide, which you can use to develop servlets and their related Web resources quickly.



RELATED CONCEPTS

JavaServer Pages
JSP Execution Monitor
WebSphere Test Environment

RELATED TASKS

Using the JSP Execution Monitor
Retrieving syntax error information
Loading generated servlet externally
Debugging JSP-generated code in VisualAge for Java
Working around problems
Samples that implement JSP and servlet technology - overview
Sample: Signs of the Zodiac
Sample: Servlet Engine Configuration
Sample: Find the Leap Years

RELATED REFERENCES

JSP debug flow

JSP/Servlet Development Environment

JavaServer Pages

JavaServer Pages (JSP), a server-side scripting technology, allows you to embed Java code within static Web pages (HTML documents), and execute the Java code when the page is served. By separating presentation logic (content presentation) from business logic (content generation), the JavaServer Pages technology makes it easy for both the Java programmer and the Web page designer to create HTML pages with dynamic content.

With JavaServer Pages, you can easily access reusable components. JSP technology allows you to use all the Java APIs available to any Java applet or application, such as Java beans, enterprise beans (EJB beans), and servlets.

The scripted HTML file has a .jsp extension, so that the server can identify it as a JSP file. Before the JSP page is served, the JSP syntax is parsed and processed into an object on the server side. The resulting object generates dynamic HTML content and sends it back to the client.

A JSP file can be directly requested as a URL, called by a servlet, or called from within an HTML page. In all three cases, the servlet engine compiles the JSP into a servlet and runs it. The compilation is performed the first time the JSP file is requested, and each time the JSP source changes. Being able to compile on demand means that you can deploy new versions of JSP files into a running Web application. As well, performance is improved because you do not have to compile, load, and run a servlet each time a request is made to the server.

In VisualAge for Java, Version 3.5, both JSP 0.91 and JSP 1.0 are supported. The default settings support JSP 1.0. For instructions on changing the settings to use JSP 0.91, see "Migrating between JSP 0.91 and JSP 1.0 support."

JSP beans
One of the most powerful features of JavaServer Pages is that you can access Java beans and EJB beans from within a .jsp file. Java beans can be class files, serialized beans, beans that are dynamically generated by a servlet, or a servlet itself.

You can do any of the following:

- Create a bean from a serialized file or a class file
- Refer to a bean from an HTTP session
- Pass a bean from the servlet to the JSP page

You can access reusable server-side components simply by declaring the components in the .jsp file. Bean properties can then be accessed inside the file.

Use the bean tag to declare a bean inside a .jsp file. Use JSP syntax and HTML template syntax to access the bean.

Dynamic content generation
Dynamic content generation works in the following way:

1. The user fills in an HTML form, and clicks Submit. This will post the request to a Java servlet.
2. The servlet reads the input parameters and passes the parameters to Java beans that perform the business logic.
3. Based on the outcome of the business logic and the user profile, the servlet calls a JSP page to present the results.
4. The JSP page extracts the results from the Java beans and merges them with the HTML page. The dynamically generated HTML page is returned to the user.

You can easily create JSP files by using WebSphere Studio tools, and text editors.

RELATED CONCEPTS

JSP and servlets
JSP Execution Monitor

RELATED TASKS

Loading the required features
Configuring DB2 to work with data-enabled samples
Migrating between JSP 0.91 and JSP 1.0 support

Using the JSP Execution Monitor
Retrieving syntax error information
Loading generated servlet externally
Debugging JSP-generated code in VisualAge for Java
Working around problems
Samples that implement JSP and servlet technology - overview

RELATED REFERENCES

JSP 1.0 programming reference
JSP 0.91 programming reference

JSP Execution Monitor

The JSP Execution Monitor allows you to monitor the execution of JSP source, the JSP-generated Java source, and the HTML output. With the JSP Execution Monitor, you can efficiently monitor JSP run-time errors. The JSP Execution Monitor displays the mapping between the JSP and its associated Java source code, and allows you to insert breakpoints in the JSP source.

If you find an error in a JSP page, you can also modify the JSP source in a text editor, and then run the JSP source in the JSP Execution Monitor. To load the updated version of the JSP source into the JSP Execution Monitor, you simply have to refresh your Web browser.

The JSP Execution Monitor highlights the location of syntax errors in both the JSP and JSP-generated Java source.

RELATED CONCEPTS

JavaServer Pages
JSP and servlets

RELATED TASKS

Migrating between JSP 0.91 and JSP 1.0 support
Using the JSP Execution Monitor
Retrieving syntax error information
Loading generated servlet externally
Debugging JSP-generated code in VisualAge for Java
Working around problems
Samples that implement JSP and servlet technology - overview

RELATED REFERENCES

JSP debug flow

Setting up the environment

Loading the required features

To work with JavaServer Pages technology in VisualAge for Java, you must first load the necessary features from the repository into the workspace.

To perform a feature install, select **File** > **Quick Start** > **Features** > **Add Features**. Then, select **WebSphere Test Environment**. Click **OK**. This automatically loads the following projects into your IDE workspace:

- IBM WebSphere Test Environment
- IBM JSP Examples

The IBM WebSphere Test Environment feature contains the WebSphere integration components, the Servlet API classes, and the JSP Execution Monitor.

RELATED CONCEPTS

JavaServer Pages
JSP and servlets

RELATED TASKS

Configuring DB2 to work with data-enabled samples
Migrating between JSP 0.91 and JSP 1.0 support

Configuring DB2 to work with data-enabled JSP applications

To work with any data-enabled JSP applications, you must have DB2 installed. You need the DB2 Universal Database Personal Edition only. This edition provides a local DB2 server.

Note: You can also work with data-enabled JSP applications using the DB2 Connect Personal Edition. If you choose to use the DB2 Connect Personal Edition only (without installing DB2 Universal Database Personal Edition), then you must be able to connect to a remote DB2 server.

If you have installed the DB2 Universal Database Personal Edition and you now want to install the DB2 Software Developer's Kit (SDK), ensure that the DB2 database server and the DB2 administration server are not running.

When you have installed the DB2 SDK, you are given three options for rebooting. Select the following option:

> **Reboot only. (Manually start the Client Configuration Manager at a later time.)**

To stop the DB2 database and administration servers, do the following:

1. On the Windows desktop, select **Start** > **Programs** > **DB2 for Windows NT/95** > **Command Window**.
2. From the command line, issue the following two commands:
   db2stop
   db2admin stop
   These commands stop the servers.
   If you are running Windows‟ 95 and you receive a message that says you are not authorized to perform this action, then log out of Windows 95 and log on using another user ID and password.

RELATED CONCEPTS

JavaServer Pages
JSP and servlets

RELATED TASKS

Loading the required features
Migrating between JSP 0.91 and JSP 1.0 support

Migrating between JSP 1.0 and JSP 0.91 support

In VisualAge for Java, Version 3.5, both JSP 0.91 and JSP 1.0 are supported. The default settings support JSP 1.0. You can change the settings to support JSP 0.91 by revising the following properties file:

X:\IBMVJava\ide\project_resources\IBM WebSphere Test Environment\hosts\default_host\default_app\servlets\default_app.webapp

where X:\IBMVJava is the VisualAge for Java installation directory. In this file, change the following:

```
...<description>JSP support servlet</description>
<code>com.ibm.ivj.jsp.runtime.JspDebugServlet</code><init-parameter>...
```

back to:

```
...<description>JSP support servlet</description>
<code>com.ibm.ivj.jsp.debugger.pagecompile.IBMPageCompileServlet</code><init-parameter>...
```

Save the file.

If you want to switch back to JSP 1.0 support, then change the following:

```
...<description>JSP support servlet</description>
<code>com.ibm.ivj.jsp.debugger.pagecompile.IBMPageCompileServlet</code><init-parameter>...
```

to:

```
...<description>JSP support servlet</description>
<code>com.ibm.ivj.jsp.runtime.JspDebugServlet</code><init-parameter>...
```

RELATED CONCEPTS

JavaServer Pages
JSP and servlets

RELATED TASKS

Loading the required features
Configuring DB2 to work with data-enabled samples
Working around problems

RELATED REFERENCES

JSP 0.91 programming reference
JSP 1.0 programming reference

Using the JSP Execution Monitor

Use the JSP Execution Monitor to monitor the execution of JSP source, JSP-generated Java source, and HTML output.

Working with JSP files in a team environment
When working with JSP files in a team environment with a shared repository, the workspace owner must also be the owner of the project that contains the generated JSP-generated servlets and class files. See “Debugging JSP-generated code in VisualAge for Java.”

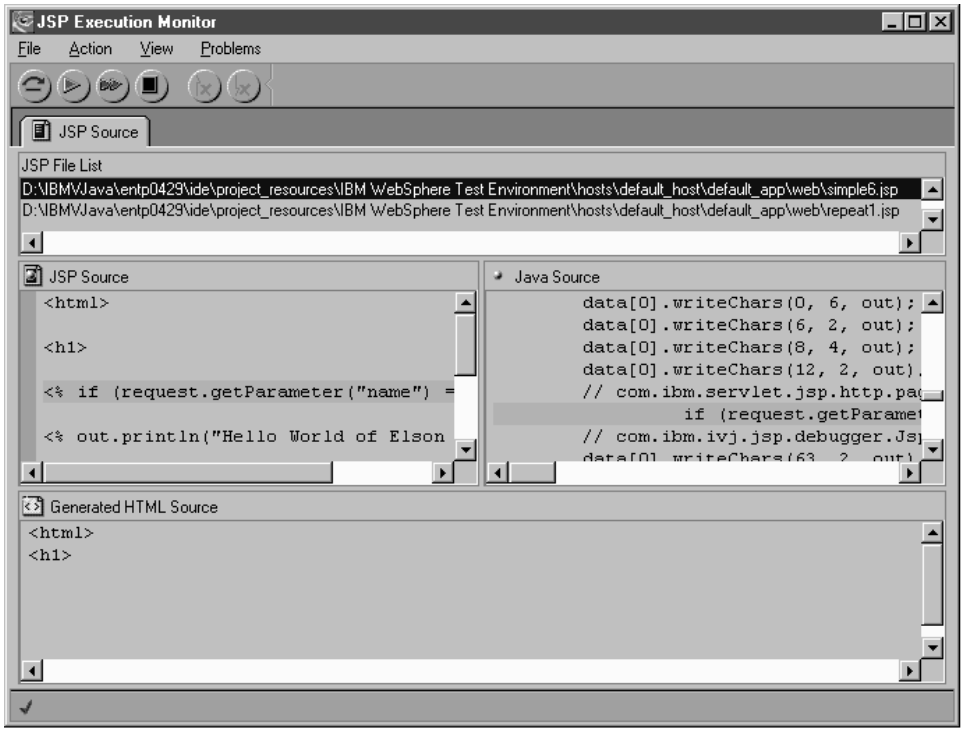
Launching the JSP Execution Monitor
To launch the JSP Execution Monitor, do the following:

1. From the **Workspace** menu, select **Tools** > **WebSphere Test Environment**. The WebSphere Test Environment Control Center dialog opens.
2. Select **JSP Execution Monitor Options**. In the JSP Execution Monitor window, complete the following:
   a. The default internal port number for the use of the JSP Execution Monitor is 8082. If port number 8082 is already in use, change the port number in the JSP Execution Monitor internal port number field. The port number must be between 1024 and 66536. **Note:** If the JSP Execution Monitor is running, you will not be able to change the port number.
   b. By default, the JSP Execution Monitor mode is disabled. You must select **Enable monitoring JSP execution** to enable the JSP Execution Monitor when a JSP file gets loaded.
   c. By default, the Retrieve syntax error information option is disabled. Selecting **Retrieve syntax error information** allows you to know exactly where a syntax error occurs in the code source.
   For more details, see Retrieving syntax error information.
   d. When you have selected your options, click **Apply**. (In this case, ensure that at least the **Enable monitoring JSP execution** option is selected, so that you can familiarize yourself with the JSP Execution Monitor window.)
3. In the WebSphere Test Environment Control Center, select **Servers** > **Servlet Engine** to specify the JSP settings.
   a. By default, the Load generated servlet externally option is disabled. Selecting **Load generated servlet externally** allows you to load a generated servlet, so that the servlet does not get imported into the IDE.
   When you select to load the generated servlet externally, you also have the option of selecting to halt the loading at the beginning of the service method. Select **halt at the beginning of the service method** if you want see the execution of the generated servlet in the IDE debugger. De-select this option if you want to be able to set breakpoints in a specific line of the generated code.
   For more details, see Loading generated servlet externally.
   b. By default, the Enable JSP source debugging option is disabled. Selecting **Enable JSP source debugging** allows you to debug JSP source using the IDE debugger. You can select this option only when the Load generated servlet externally option is selected.
   For more details, see Enabling JSP source debugging.
4. Launch the WebSphere Test Environment Servlet Engine. In the WebSphere Test Environment Control Center, select **Servlet Engine**. In the Servlet Engine window, click **Start the Servlet Engine**. Check the status line messages in the Control Center to ensure that the Servlet Engine is running in VisualAge for Java.
   Once you have the WebSphere Test Environment running in VisualAge for Java, you can serve JSP files and HTML files from the designated document root. See "Starting, stopping, and configuring WTE services."
5. Then, launch a JSP file in the Web browser, using the default WebSphere port:8080.

The JSP Execution Monitor window has four panes:

1. The JSP File List pane lists all the JSP files that you have launched in your browser.
2. The JSP Source pane displays the JSP source code.
3. The Java Source pane displays the JSP-generated Java source code.

4. The Generated HTML Source pane displays the run-time generated HTML source code.



Monitoring the execution of JSP source
Each time you load a JSP file in your browser, the file name is displayed in the JSP File List pane. When the JSP file has completed loading in the JSP Execution Monitor, click the file name in the JSP File List pane to display the JSP source code in the JSP Execution Monitor. In the JSP Source pane, you can step through the JSP source by clicking the Step

button in the tool bar at the top of the screen, or by selecting **Action** > **Step**. Click the Step

button to step through each significant block of JSP code in the JSP source.

The Generated Java Source pane displays the generated Java code of the highlighted block of JSP code in the JSP Source pane. The Java source that is highlighted in the Generated Java Source pane is the currently executed JSP source as indicated in the JSP Source pane. The Generated HTML Source pane displays the generated HTML code of the highlighted block of JSP code in the JSP Source pane.

You can also set breakpoints in the JSP source code by double-clicking in the left margin (dark gray area) of the JSP Source pane. After you set your breakpoints, click the Run

button, or select **Action** > **Run**, to execute the JSP source code up to the set breakpoint. Double-click a breakpoint again to remove it. You cannot set a breakpoint on an empty line. If you want to execute the JSP source code to the end *without* stepping through or highlighting each significant block of JSP code, click the Fast Forward

button, or select **Action** > **Fast Forward**.

When you select **Run** or **Fast Forward**, the other push buttons are still enabled. Therefore, when you select **Run** or **Fast Forward** while the JSP code is running, you can select **Step** to pause the program at any point. Also, while in the **Run** mode, you can switch to the **Fast Forward** mode, and vice versa.

When the code has been fully executed, the **Step**, **Run**, **Fast Forward**, and **Terminate** buttons (along with **Step**, **Run**, and **Terminate** in the **Action** drop-down menu) are no longer available. Select **File** > **Exit** to close the JSP Execution Monitor.

You can click Terminate

to complete code execution.

To maximize, minimize, or detach a pane, click the pane in order to select it, and then select the appropriate item from the **View** menu.

Disabling the JSP Execution Monitor
If you want to simply run the JSP source without monitoring the execution of the source itself, then you must disable the JSP Execution Monitor. Deselect **Enable monitoring JSP execution** in the JSP Execution Monitor Options page of the WebSphere Test Environment Control Center. See the file “Starting, stopping, and configuring WTE services” in the online help.

When the JSP Execution Monitor is disabled, you can debug the generated servlet in the IDE. You will find the generated servlet under the JSP Page Compile Generated Code project in your workspace. See “Debugging JSP-generated code in VisualAge for Java.”

RELATED CONCEPTS

JSP Execution Monitor
JavaServer Pages
JSP and servlets

RELATED TASKS

Starting, stopping, and configuring WTE services
Retrieving syntax error information
Loading generated servlet externally
Loading the required features
Configuring DB2 to work with data-enabled samples
Migrating between JSP 0.91 and JSP 1.0 support
Debugging JSP-generated code in VisualAge for Java
Working around problems
Samples that implement JSP and servlet technology - overview

RELATED REFERENCES

JSP debug flow
Generated servlet file names

Loading generated servlet externally

In VisualAge for Java, Version 3.0, each time a JSP file is compiled, the generated servlet always gets imported into the IDE. This occurs so that you can debug the generated servlet using the IDE debugger. However, the size of the repository increases significantly when JSP files are frequently changed and launched (because the servlet is getting recompiled each time). To prevent the unnecessary size increase in the repository, we have added an option that allows you to load the generated servlet externally. This way, the generated servlet does not get imported into the IDE.

If you select the Load generated servlet externally option in the Servlet Engine page of the WebSphere Test Environment Control Center, and then request a JSP file, then the pagecompiler will check to see if the generated servlet already exists in the workspace. If the servlet class exists, then a different file naming convention for the generated servlet class will be used, so that the server will not pick up a previous version of the generated servlet.

The generated servlet is compiled into a class file when the Load generated servlet externally option is enabled. Consequently, the generated servlet will not be imported into the IDE.

When the Load generated servlet externally feature is enabled, the Retrieve syntax error information option is also enabled.

To enable the Load generated servlet externally feature, complete the following:

1. Under the **Workspace** menu in the IDE workbench, select **Workspace** > **Tools** > **WebSphere Test Environment**. The WebSphere Test Environment Control Center opens.
2. Select **JSP Execution Monitor Options**.
3. By default, the Load generated servlet externally option is disabled. Selecting **Loading generated servlet externally** allows you to load a generated servlet, so that the servlet does not get imported into the IDE.
   When you select to load the generated servlet externally, you also have the option of selecting to halt the loading at the beginning of the service method.
4. Select **halt at the beginning of the service method** if you want to see the execution of the generated servlet in the IDE debugger.

For a detailed description on debugging external .class files in the IDE debugger, please refer to the VisualAge for Java Integrated Debugger online documentation.

Debugging the generated servlet class file
The default JSP cache directory is automatically added to the Servlet Engine classpath. This allows you to debug the generated servlet class files using the IDE debugger.

- For JSP 0.91, the default JSP cache directory is X:\IBMVJava\Ide\project_resources\IBM WebSphere Test Environment\temp\default_app
- For JSP 1.0, the default JSP cache directory is X:\IBMVJava\Ide\project_resources\IBM WebSphere Test Environment\temp\JSP1_0\default_app

We recommend that you accept the above default directory settings. However, if you want to modify the directory settings, you must do so in the default_app.webapp file. The default_app.webapp file is located in

X:\IBMVJava\Ide\project_resources\IBM WebSphere Test Environment\hosts\default_host\default_app\servlets

In all of the above cases, X:\IBMVJava is the directory in which you installed VisualAge for Java.

Each generated servlet class file contains a number extension in the file name; the number extension changes each time the servlet is recompiled. For example, with only the load generated servlet externally option selected, a JSP file called simple.jsp will have a generated servlet name called _simple_xjsp_2079706272.java where 2079706272 is a random number.

RELATED CONCEPTS

JSP Execution Monitor
JavaServer Pages
JSP and servlets

RELATED TASKS

Using the JSP Execution Monitor
Retrieving syntax error information
Debugging JSP-generated code in VisualAge for Java
Working around problems

Enabling JSP source debugging

Use the WebSphere Test Environment Control Center to enable the JSP source debugging feature. The JSP Execution Monitor's Enable JSP source debugging feature allows you to debug JSP source using the IDE debugger.

When you select to enable JSP source debugging, and then load a JSP file in your browser, the JSP source is displayed in the IDE debugger window. You can then debug the JSP source by setting breakpoints in the source.

For detailed instructions on debugging in the IDE, see the VisualAge for Java online help.

RELATED CONCEPTS

JSP and servlets
JavaServer Pages
JSP Execution Monitor

RELATED TASKS

Using the JSP Execution Monitor
Loading generated servlet externally
Retrieving syntax error information

Retrieving syntax error information

The JSP Execution Monitor's syntax error checking feature allows you to know exactly where a syntax error occurs in the code source.

To set the JSP Execution Monitor to retrieve syntax error information, complete the following:

1. Under the Workspace menu in the IDE workbench, select **Workspace** > **Tools** > **WebSphere Test Environment**. The WebSphere Test Environment Control Center opens.
2. Select **JSP Execution Monitor Options**.
3. By default, the JSP Execution Monitor is *not* set to display the exact syntax error when an error occurs. Select **Retrieve syntax error information** if you want to know not only that a syntax error exists, but also where that syntax error occurs in the code source.

   If the JSP Execution Monitor is not enabled, then the syntax errors are displayed only in the browser. When you enable the JSP Execution Monitor, the syntax errors will be visually mapped back to the JSP source and displayed in the JSP Execution Monitor.

   If you do not set this feature and syntax errors exist in the JSP file being loaded, then the JSP Execution Monitor will not launch even if the JSP Execution Monitor is enabled. Instead, the browser will display a compile error message, “Error getting compiled page”.

   Click **Apply** to save.
4. The JSP Execution Monitor will list the error message in the status line located at the bottom of the JSP Execution Monitor window. The error will be listed as a JSP or Generated Java syntax error. If the error is a JSP syntax error, the generated Java source will not be displayed in the JSP Execution Monitor (a JSP source containing a syntax error cannot properly generate Java source). If the error is a Java syntax error, the JSP Execution Monitor will display both the JSP source and the JSP-generated Java source. The syntax error will be highlighted in both the JSP source and the Java source.
   JSP syntax errors that have not been picked up by the pagecompiler preprocessor will be displayed as Generated Java syntax errors.
   To step from one syntax error to another, click the Previous Problem

   or Next Problem

   buttons, or select **Problems** > **Previous Problem** or **Problems**> **Next Problem**.

Clear the **Retrieve syntax error information** check box once you have fixed all the syntax errors. This will result in faster compile time, and will allow you to focus on run-time errors.

Debugging JSP source
In general, when working with a JSP file, complete the following steps for debugging JSP source efficiently:

1. In the JSP Execution Monitor Option dialog box, clear both the **Enable monitoring JSP execution** and **Retrieve syntax error information** check boxes. When you load the file, and the browser displays “Error getting compiled page”, it means that a syntax error exists in the JSP file.
2. Then, select both **Enable monitoring JSP execution** and **Retrieve syntax error information**.
3. Reload the JSP file in your browser in order to display the syntax error in the JSP Execution Monitor.
4. Modify the JSP file, correcting the syntax errors.

5. Reload the JSP file in your browser.
6. Repeat these steps until no syntax error remains in the JSP file.

If you run the JSP file to completion and discover that the dynamic content of the results is not correct, then select **Enable monitoring JSP execution** and deselect **Retrieve syntax error information**. Then, reload the JSP file in the browser in order to run the JSP file in the JSP Execution Monitor.

See "Using the JSP Execution Monitor."

RELATED CONCEPTS

JSP Execution Monitor
JavaServer Pages
JSP and servlets

RELATED TASKS

Using the JSP Execution Monitor
Loading generated servlet externally
Debugging JSP-generated code in VisualAge for Java
Working around problems

Debugging JSP-generated servlet code in VisualAge for Java

Because VisualAge for Java can run servers in the IDE, VisualAge for Java allows you to debug servlets in the IDE using the VisualAge for Java debugger. In the IDE, you can set breakpoints in a servlet, just as you can for applets. You can debug JSP-generated code in the IDE by setting breakpoints in the code source. You can cut and paste the changes that you made to the JSP-generated code (Java code) back into the JSP source.

When you run the servlet from a browser, the breakpoints halt the servlet code, and a Debugger window opens, which allows you to step through and edit the code. When you resume running the servlet, your code changes are immediately reflected in any output to the browser. This allows you to do real-time debugging of the servlet code while running your application.

When JSP-generated code is imported into the IDE, the following occurs:

1. The JSP source is fed to a page compiler, which creates an executable object (for example, a Java HTTP servlet).
2. VisualAge for Java then imports the generated servlet code into the following default directory:
   X:\IBMVJava\IDE\project_resources\IBM WebSphere Test Environment\temp\default_app
   where X:\IBMVJava is the directory in which VisualAge for Java is installed. You can run and debug the servlet by using your browser to call the JSP file that created the servlet.

Attention: For JSP 0.91, if the JSP Execution Monitor is disabled, then you will find the generated code in the class *_sourcename*_xjsp under the pagecompile package in the JSP Page Compile Generated Code project. If the JSP Execution Monitor is enabled, then you will find the generated code in the class *_sourcename*_xjsp_debug under the pagecompile package in the JSP Page Compile Generated Code project.

The generated code in the class *_sourcename*_xjsp_debug contains extra internal debug information. When working with the business logic of a JSP source, we

recommend that you disable the JSP Execution Monitor and work with the generated code in the class *_sourcename*_xjsp. This code does not contain extra internal debug information.

JSP 'Hello World' sample
Use the following JSP sample to walk through the steps of debugging JSP-generated code in VisualAge for Java. The sample performs some arithmetic operations, contains a loop, and has a number of places where you can set breakpoints, and examine field and object values.

```
<html>
<body>
<h2>This is a JavaServer Pages (JSP) example using a 'Scriplet' in which you can step
through the generated Java code in VisualAge for Java, Version 3.0</h2>

<% int sum = 0;%>
<% int number = 123;%>
<% int counter = 1;%>
<% int loops = 20;%>
<% out.println("Total number of loops to process: " + loops);%>
<% // try setting a breakpoint in the for loop below and step through the debugger %>
<% for (; counter < loops; counter++) {%>
<% out.println("<li>Hello World from VisualAge for Java, Version 3.0");%>
<% out.println("<li>Current loop number is: " + counter);%>
<% sum = number + counter;%>
<% out.println("<li>" + counter + " + the number " + number + " is = " + sum);%>
<% }%>
<h3>That was dynamic output from a JavaServer Page</h3>
</body>
</html>
```

For detailed instructions on debugging in the IDE, see the VisualAge for Java online help.

RELATED CONCEPTS

JavaServer Pages
JSP and servlets

RELATED TASKS

Using the JSP Execution Monitor
Loading generated servlet externally
Working around problems
Samples that implement JSP and servlet technology - overview

Working around problems

This topic contains information on troubleshooting problems that you might encounter while working with JavaServer Pages in VisualAge for Java. Use the following list to identify a specific problem, then see the associated section that addresses the problem, following the list.

- "I cannot run an external servlet inside VisualAge for Java", or "I am having problems running a JSP file that calls a servlet."
  See Running external servlets inside VisualAge for Java (page 15).
- "I am getting a Page Compile error message each time I load a JSP file in my browser."
  See Resolving case-sensitivity issue (page 15).

- "I am not able to serve anything — HTML files, JSP files, and servlets — from my browser."
  See Serving files from browser (page 16).
- "I have detached the entire JSP Execution Monitor pane."
  See Restoring the entire JSP Execution Monitor pane (page 16).
- "I have altered the browser preferences for a detached JSP Execution Monitor pane, and now I have lost the VisualAge for Java, Version 3.0 look and feel."
  See Restoring the VisualAge for Java, Version 3.0 look and feel to the JSP Execution Monitor (page 16).
- "The debugger opens when I load a JSP file."
  See Preventing the debugger from opening (page 16).
- "I cannot execute multiple JSP files at a time in the JSP Execution Monitor."
  See Executing multiple JSP files at one time (page 16).
- "I cannot reload an older version of a JSP file."
  See Reverting to an older version of a JSP file (page 17).
- "I cannot use the JSP Execution Monitor to step into script tag."
  See Debugging code inside a script tag (page 17).
- "I cannot re-detach a pane in the JSP Execution Monitor."
  See Re-detaching a pane in the JSP Execution Monitor (page 17).
- "I have removed all panes from the JSP Execution Monitor, and now have problems detaching into a new window."
  See Detaching into new window (page 17).

Running external servlets inside VisualAge for Java
To run an external servlet inside VisualAge for Java, you must first copy the servlet's class files into the following directory:

```
X:\IBMVJava\IDE\project_resources\IBM WebSphere Test
Environment\hosts\default_host\default_app\servlets\
```

where X:\IBMVJava is the directory in which VisualAge for Java is installed.

After you have copied the files, start the Servlet Engine from the WebSphere Test Environment Control Center.

Similarly, to run a Java bean contained in a JSP file, you must first copy the bean into the same directory.

To debug the code, complete the above steps, and import the Java bean or servlet code into the VisualAge for Java IDE.

Resolving case-sensitivity issue
When loading a JSP file in your browser, the URL that you qualify must match the file structure of your document root. Because the WebSphere Application Server page compiler creates case-sensitive package names, you must use consistent case when specifying URLs of JSP files. Although case-sensitivity is not an issue on Windows NT, Java itself is case sensitive, so when the page compiler creates a package name from the directory specified in the URL, the case of the file names is relevant in VisualAge for Java.

For example, if the JSP subdirectory name in your document root directory is in uppercase, enter the following URL when loading the JSP samples:

http://localhost:8080/JSP/index.html

However, if the 'JSP subdirectory' name in your document root directory is in lowercase, enter the following URL when loading the JSP samples:

http://localhost:8080/jsp/index.html

WIN

Note: On Windows NT, if the first letter of a directory/file name appears in uppercase, then it means that the rest of the file name is also 'hard-coded' as uppercase.

Serving files from browser
If you are not able to serve any files - HTML, JSP, or servlets - from your browser, then shut down Servlet Engine and then restart it (using the WebSphere Control Center). When you have restarted Servlet Engine, try launching your files again. See the file "Starting, stopping, and configuring WTE services" in the online help.

Restoring the entire JSP Execution Monitor pane
In the JSP Execution Monitor, if you double-click the main tab labeled **JSP Source**, the entire JSP Execution Monitor window will be detached. To restore the window, close the detached window by clicking the close **x** button in the top right-hand corner of the window.

Restoring the VisualAge for Java, Version 3.0 look and feel to the JSP Execution Monitor
When you detach a JSP Execution Monitor pane into a new window, you have the option of changing browser preferences by selecting **File** > **Preferences**. Once you are in the Browser Preferences dialog, you have the option of changing the appearance fields (by selecting **General** > **Appearance**). Under **Appearance**, in the **Family** entry field, select **Visual Age 3.0**, if it is not already selected. In the **Platform** entry field, select **Windows**, if it is not already selected. These settings will allow you to restore the VisualAge for Java, Version 3.0 look and feel to the JSP Execution Monitor.

If you changed any of the preferences in the detached pane and you want to restore the JSP Execution Monitor to its original look and feel, do the following:

1. Reverse the changes while working with the same settings that you had previously set (for example, if you previously set the Family to be Granite, you must reverse the changes with the Family set as Granite).
2. Then, in the detached pane, select **File** > **Preferences** > **General** > **Appearance**, and select **Visual Age 3.0** in the Family entry field, and then select **Windows** in the Platform entry field.

Preventing the debugger from opening
In the JSP Execution Monitor Option dialog, you have the option of enabling or disabling **Retrieve syntax error information**. If you disable **Retrieve syntax error information**, the debugger window will open when you launch a JSP file that implements an interface that does not exist in VisualAge for Java. When **Retrieve syntax error information** is disabled, the compile-time error is not detected. An attempt to load the class will fail, which will result in the debugger window opening.

To prevent the debugger from opening, enable **Retrieve syntax error information**. Then, when you launch the JSP file, the error will be reported as expected.

Executing multiple JSP files at one time
You can run multiple JSP files at a time in the JSP Execution Monitor. However, when you are in the process of executing one JSP file, and you go to run a second JSP file, the previous JSP file will stop executing while you execute the subsequent JSP file. To continue executing the previous JSP file, you must go back to it by clicking that file in the JSP File List pane. When the file is displayed in the JSP Execution Monitor, click the **Run** button to continue executing the previous JSP file.

Reverting to an older version of a JSP file
The WebSphere Application Server will recompile a JSP file only when the file's time stamp is later than the file in the cache. If you make a backup copy of a JSP file, and later need to revert to this backup copy, you must update the JSP file's last modified time to the present time.

To update the file's last modified time to the present time, edit the file and resave it.

Debugging code inside a script tag
The JSP Execution Monitor does not allow you to step into script tag.

To debug the code contained within a script tag, simply set breakpoints in the code in the IDE debugger.

Re-detaching a pane in the JSP Execution Monitor
When you detach a pane in the JSP Execution Monitor window (for example, the HtmlView pane), and then close it, you will not be able to detach the pane a second time.

To work around this problem, you must shut down, and restart the JSP Execution Monitor.

Detaching into new window
When you remove all of the panes from the JSP Execution Monitor so that the window is empty, and then select **Detach into new window**, the last pane you removed will disappear and cannot be replaced.

To work around this problem, you must shut down, and restart the JSP Execution Monitor.

RELATED CONCEPTS

JavaServer Pages
JSP Execution Monitor
JSP and servlets

RELATED TASKS

Migrating between JSP 0.91 and JSP 1.0 support
Using the JSP Execution Monitor
Debugging JSP-generated code in VisualAge for Java

Chapter 2. Samples that implement JSP and servlet technology - overview

The following three samples provide a hands-on introduction to JavaServer Pages and servlet technology:

- **Signs of the Zodiac Sample** This servlet samples determines the zodiac sign for the inputted date. An HTTP servlet is used to generate dynamic output content based on input from the HTML page. Use this sample to familiarize yourself with debugging servlets using VisualAge for Java.
- **Servlet Engine Configuration Sample** This servlet sample displays servlet engine configuration information, such as supported transports, virtual hosts, system properties, and so forth. Use this sample to familiarize yourself with the environment variables.
- **Find the Leap Years Sample** This JSP/servlet sample determines the next 10 leap years from and including the Start Year. A JSP file is used to generate dynamic content, so that the application logic is separated from the Web page design. Use this sample to familiarize yourself with using the Create Servlet SmartGuide and the JSP Execution Monitor.
  Attention: This sample was written using JSP 1.0. If you are using JSP 0.91, you can modify the method performTask() in the servlet class com.ibm.ivj.wte.samples.leapyear.LeapYear to call LeapYearResults091.jsp instead of LeapYearResults.jsp.

To work with JavaServer Pages technology in VisualAge for Java, you must first load the necessary features from the repository into the workspace. To perform a feature install, select **File** > **Quick Start** > **Features** > **Add Features**. Then, select **WebSphere Test Environment**. Click **OK**. This automatically loads the following projects into your IDE workspace:

- IBM WebSphere Test Environment
- IBM JSP Samples

The IBM WebSphere Test Environment feature contains the WebSphere integration components, the Servlet API classes, and the JSP Execution Monitor.

You can now begin to work with the Signs of the Zodiac, Servlet Engine Configuration, and Find the Leap Years samples.

RELATED CONCEPTS

JavaServer Pages
JSP Execution Monitor
JSP and servlets

RELATED TASKS

Sample: Signs of the Zodiac
Sample: Servlet Engine Configuration
Sample: Find the Leap Years
Loading the required features
Configuring DB2 to work with data-enabled samples
Migrating between JSP 0.91 and JSP 1.0 support
Using the JSP Execution Monitor
Debugging JSP-generated code in VisualAge for Java

Sample: Signs of the Zodiac

The Signs of the Zodiac sample determines the zodiac sign for the inputted date. An HTTP servlet is used to generate dynamic output content based on input from the HTML page. Use this sample to familiarize yourself with debugging servlets using VisualAge for Java.

To access the samples index page, titled VisualAge for Java Servlet & JSP Samples, already installed on your hard drive, do the following:

1. Launch VisualAge for Java.
2. Launch the WebSphere Test Environment Servlet Engine. In the WebSphere Test Environment Control Center, select **Servlet Engine**. In the Servlet Engine window, click **Start the Servlet Engine**. Check the status line messages in the Control Center to ensure that the Servlet Engine is running in VisualAge for Java.
   Once you have the WebSphere Test Environment running in VisualAge for Java, you can serve JSP files and HTML files from the designated document root. See "Starting, stopping, and configuring WTE services."
3. Start your Web browser.
4. Load the JSP Samples index page in your Web browser, by entering the following URL:
   http://localhost:8080/JSP/index.html
   This page contains links to three JSP samples, including the Signs of the Zodiac servlet sample.
5. Click **Look Up Your Sign**.
6. Select a Day and Month, and click **Look Up Your Sign**. The browser displays the appropriate zodiac sign as generated by the servlet.

To debug the servlet:

1. In the Workbench, expand the LookupSign class (it is located in the IBM JSP Examples project under the com.ibm.ivj.wte.samples.signs package).
2. Set a breakpoint in the doGet method at the following line:
   ```
   response.setContentType("text/html");
   ```
3. Reload the servlet (following the instructions above).
4. The Debugger will automatically launch, running the servlet to the set breakpoint.
5. Step through the servlet, and examine the variables.
6. You can also modify the code, while debugging it. For example, remove the comment tags from the following lines:
   ```
   // Calendar cal=Calendar.getInstance();
   // cal.setTime(new Date());
   // out.println("<H2>The sign for today is " +
   getSign(cal.get(Calendar.MONTH)+1, cal.get(Calendar.DAY_OF_MONTH)) +
   "</H2><P>");
   ```
7. Save the changes.
8. The servlet will be rerun.
9. Click the Step Over button to execute the newly revised lines.
10. The sign for today's date is now displayed in the browser.
11. Click the Resume button to resume the execution of the servlet.

RELATED CONCEPTS

JavaServer Pages
JSP Execution Monitor
JSP and servlets

RELATED TASKS

Samples that implement JSP and servlet technology - overview
Sample: Servlet Engine Configuration
Sample: Find the Leap Years
Loading the required features
Using the JSP Execution Monitor
Debugging JSP-generated code in VisualAge for Java

Sample: Servlet Engine Configuration

The Servlet Engine Configuration servlet sample displays servlet engine configuration information, such as supported transports, virtual hosts, system properties, and so forth. Use this sample to familiarize yourself with the environment variables.

To access the samples index page, titled VisualAge for Java Servlet & JSP Samples, already installed on your hard drive, do the following:

1. Launch VisualAge for Java.
2. Launch the WebSphere Test Environment Servlet Engine. In the WebSphere Test Environment Control Center, select **Servlet Engine**. In the Servlet Engine window, click **Start the Servlet Engine**. Check the status line messages in the Control Center to ensure that the Servlet Engine is running in VisualAge for Java.
   Once you have the WebSphere Test Environment running in VisualAge for Java, you can serve JSP files and HTML files from the designated document root. See "Starting, stopping, and configuring WebSphere Test Environment services" in the online help.
3. Start your Web browser.
4. Load the JSP Samples index page in your Web browser, by entering the following URL:
   http://localhost:8080/JSP/index.html
   This page contains links to three JSP samples, including the Servlet Engine Configuration servlet sample.
5. Click **Display the Servlet Engine Configuration Information**.
6. The servlet engine configuration information is now displayed in your browser. Take a look at the pages to familiarize yourself with the environment variables.

RELATED CONCEPTS

JavaServer Pages
JSP Execution Monitor
JSP and servlets

RELATED TASKS

Samples that implement JSP and servlet technology - overview
Sample: Signs of the Zodiac
Sample: Find the Leap Years
Loading the required features
Using the JSP Execution Monitor
Debugging JSP-generated code in VisualAge for Java

Sample: Find the Leap Years

The Find the Leap Years JSP/servlet sample determines the next 10 leap years from and including the inputted Start Year. A JSP file is used to generate dynamic content, so that the application logic is separated from the Web page design. Use this sample to familiarize yourself with using the Create Servlet SmartGuide and the JSP Execution Monitor.
Note: This sample was written using JSP 1.0. If you are using JSP 0.91, you can modify the LeapYear.java servlet to call LeapYearResults091.jsp instead of LeapYearResults.jsp.

To access the index page for the IBM WebSphere JSP Execution Monitor Samples, do the following:

1. Launch VisualAge for Java.
2. Launch the WebSphere Test Environment Servlet Engine. In the WebSphere Test Environment Control Center, select **Servlet Engine**. In the Servlet Engine window, click **Start the Servlet Engine**. Check the status line messages in the Control Center to ensure that the Servlet Engine is running in VisualAge for Java.
   Once you have the WebSphere Test Environment running in VisualAge for Java, you can serve JSP files and HTML files from the designated document root. See "Starting, stopping, and configuring WTE services."
3. Start your Web browser.
4. Load the JSP Samples index page in your Web browser, by entering the following URL:
   http://localhost:8080/JSP/index.html
   This page contains links to three JSP samples, including the Find the Leap Years JSP/servlet sample.
5. Click **Find the Leap Years**.
6. Enter a year, and click **Submit**. The next 10 leap years are now displayed by LeapYearResults.jsp.

To recreate the servlet using the Create Servlet SmartGuide (note that the Create Servlet SmartGuide only generates JSP 1.0 files):

1. In the Workbench, create a new project called *Servlet Sample*.
2. In this project, create a package called *servlet.sample*.
3. Copy and paste the Java bean LeapYearBean.java from the package com.ibm.ivj.wte.samples.leapyear into the servlet.sample package.
4. Click the Create Servlet button to launch the Create Servlet SmartGuide.
5. Using the SmartGuide, do the following:
   a. In the Project field, enter *Servlet Sample*.
   b. In the Package field, enter *servlet.sample*.
   c. In the Class name field, enter *LeapYear*.
   d. In the Superclass field, ensure that *javax.servlet.http.HttpServlet* is entered.
   e. Select **Import Java bean**.
   f. Click **Next**.
   g. Select the Java bean, and complete the following:
      1) In the Java bean class field, select **Browse** to specify *LeapYearBean*.
      2) In the Java bean name field, ensure that *leapYearBean* is specified.
      3) For Scope, select **request**.
   h. Click **Next**.

i. Click **Add** to add fields to the input page. Select **(int)startYear**.
j. Highlight the (int)startYear field, and click **Properties**.
k. Change the caption to *Start Year*.
l. Change the size and the maximum length to 4, and click **OK**.
m. Click **Add** to add fields to the results page. Select **(int)startYear** and **(int[])leapYears**.
n. Highlight the (int)startYear field, and click **Properties**.
o. Change the caption to *Start Year*, and click **OK**.
p. Highlight the (int[])leapYears field, and click **Properties**. Change the caption to *Leap Years*, and click **OK**.
q. Click **Add** to add methods. Select **void findLeapYears()**.
r. Click **Next**.
s. Click **Finish**.
t. The LeapYearInput.html, LeapYear.java, and LeapYearResults.jsp files are successfully generated. The servlet (.java file) is located in the IDE under the package servlet.sample, and the HTML and JSP files are located in the following directory:
X:\IBMVJava\IDE\project_resources\IBM WebSphere Test Environment\hosts\default_host\default_app\web\
where X:\IBMVJava is the directory in which VisualAge for Java is installed.

To monitor the execution of the JSP source:

1. To step through the JSP source, use the WebSphere Test Environment Control Center to enable the JSP Execution Monitor (by default, the JSP Execution Monitor mode is disabled). To launch the WebSphere Test Environment Control Center, select **Workspace** > **Tools** > **WebSphere Test Environment**.
   a. Select **JSP Execution Monitor Options**.
   b. In the JSP debug settings window, select **Enable monitoring JSP execution** and then click **Apply** to enable the JSP Execution Monitor when a JSP file gets loaded. Click **Cancel** to return to the previous mode. Select **No** when prompted to save, and then select the JSP Execution Monitor Options page again.
   For further details, see "Using the JSP Execution Monitor."
2. Go back to the JSP Samples index page, and reload the Find the Leap Years sample (following the steps above). This time, the JSP Execution Monitor will launch, displaying the source.

Attention: When the JSP Execution Monitor is enabled, you will find the generated code in the package pagecompile._JSP_debug._*xxx*_debug under the project JSP Page Compile Generated Code on your Project workspace. If the JSP Execution Monitor is disabled, you will find the generated code in the package pagecompile._JSP._*xxx*.

The generated code in the package pagecompile._JSP_debug._*xxx*_debug contains extra internal debug information. When working with the business logic of a JSP source, we recommend that you disable the JSP Execution Monitor, and work with the generated code in the package pagecompile._JSP._*xxx*. This code does not contain extra internal debug information.

See "Debugging JSP-generated code in VisualAge for Java."

RELATED CONCEPTS

JavaServer Pages
JSP Execution Monitor
JSP and servlets

RELATED TASKS

Samples that implement JSP and servlet technology - overview
Sample: Signs of the Zodiac
Sample: Servlet Engine Configuration
Loading the required features
Migrating between JSP 0.91 and JSP 1.0 support
Using the JSP Execution Monitor
Debugging JSP-generated code in VisualAge for Java

Chapter 3. JSP/Servlet Development Environment reference

JSP 0.91 programming reference

JSP tags for variable data
The JSP tags for variable data are IBM extensions to JSP 0.91. These tags enable 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 is returned to the browser.

The JSP tags for variable data are:

- <INSERT> tags for embedding variables in a JSP
- <REPEAT> tags for repeating a block of HTML tagging that contains the <INSERT> tags and the HTML tags for formatting content

  Attention: The Application Server Version 1.x supported an additional tag, <REPEATGROUP> for repeating a block of HTML tagging for data that is already logically grouped in the database. Because this release does not support the <REPEATGROUP> tag, remove that tag from any JSP files that you want to use on the Application Server Version 3.0.

These tags are designed to pass intact through HTML authoring tools. Each tag has a corresponding end tag. Each tag is case-insensitive, but some of the tag attributes are case-sensitive.

The <INSERT> tag is the base tag for specifying variable fields. The general syntax is:

```
<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 attribute is case-sensitive and cannot be used with the bean and property attributes.
- **requestattr** The attribute to access within the request object. The attribute would have been set using the setAttribute method. This attribute is case-sensitive and cannot be used with the bean and property attributes.
- **bean** The name of the JavaBean declared by a <BEAN> tag within the JSP file. See “Accessing JavaBeans” for an explanation of the <BEAN> tag. The value of this attribute is case-sensitive.
  When the bean attribute is specified but the property attribute is not specified, the entire bean is used in the substitution. For example, if the bean is type String and the property is not specified, the value of the string is substituted.
- **property** The property of the bean to access for substitution. The value of the attribute is case-sensitive and is the locale-independent name of the property. This attribute cannot be used with the requestparm and requestattr attributes.
- **default** An optional string to display when the value of the bean property is null. If the string contains more than one word, the string must be enclosed within a pair of double quotes (such as “HelpDesk number”). The value of this attribute is case-sensitive. If a value is not specified, an empty string is substituted when the value of the property is null.

Some examples are:

```
<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 the property 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 indexed properties. The optional index can be a constant (such as 2) or an index like the one described in the <REPEAT> tag section. 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>
```

The 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> tags to enclose the HTML tag in which substitution is to take place.
2. Specify the bean and property attributes:
   - To specify the bean and property attributes, use the form:
     `$(bean=`*b* `property=`*p* `default=`*d*
     where *b*, *p*, and *d* are values as described for the <INSERT> tag.
   - To specify the requestparm attribute, use the form
     `$(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.

Some examples of the alternate syntax are:

```
<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>
```

<REPEAT> tag

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 syntax of the <REPEAT> tag is:

```
<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 default is 0.
- **end** An optional ending index value for this repeat block. The maximum value is 2,147,483,647. If the value of the end attribute is less than the value of the start attribute, the end attribute is ignored.

The 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>
```

Notes:

- 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"
</dbconnect>
<dbquery id="dynamic" connection="conn" >
  Select * from myTable;
</dbquery>
<repeat index=abc>
  <insert bean="dynamic"
    property="col1(abc)">
  </insert>
</repeat>
```

Notes:

- 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. The examples produce the same output if all indexed properties have 300 or fewer elements. If there are more than 300 elements, Examples 1 and 2 will display all elements, while Example 3 will show only the first 300 elements.

Example 1 shows implicit indexing with the default start and default end index. 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 starting index. Although the index attribute is specified, the indexed property city 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 separately indexed. This capability is useful for interleaving properties on two beans, or properties that have sub-properties. In the example, two <REPEAT> blocks are nested to display the list of songs on each compact disc 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>
```

JSP tags for database access

The Application Server Version 3 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 scope of all of the tags is the Web page (the JSP) in which they are embedded. Therefore, identifiers and other tag data can be accessed only within the page.

<DBCONNECT> tag

Use the <DBCONNECT> tag to specify information needed to make a connection to a JDBC or an ODBC database. However, 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.

The <DBCONNECT> tag syntax is:

```
<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 3.0 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"
  ```

  Note: 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.

  Note: 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 via 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.

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.

<USERID> and <PASSWD> tags

Instead of hardcoding the user ID and password in the <DBCONNECT> tag, you can 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 (such as the Employee.jsp example) that requests the database connection.

The <USERID> and <PASSWD> tags must be used within a <DBCONNECT> tag. The syntax of the <USERID> and <PASSWD> tags is:

```
<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** This tag 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 tag 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.

<DBQUERY> tag
Use the <DBQUERY> tag to establish a connection to a database, submit database queries, and return the results set.

The <DBQUERY> tag:

- 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.
- Establishes a new connection
- Retrieves and caches data in the results object
- Closes the connection (releases the connection resource)

The <DBQUERY> tag syntax is:

```
<!– 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.

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>
```

<DBMODIFY> tag

Use the <DBMODIFY> tag to establish a connection to a database and then add records to a database table.

The <DBMODIFY> tag:

- 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.
- Establishes a new connection.
- Updates a table in the database.
- Closes the connection (releases the connection resource).

The <DBMODIFY> tag syntax is:

```
<!– 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>
```

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. These APIs provide a way to separate content generation (business logic) from the presentation of the content (HTML formatting). This separation enables servlets to generate content and store the content (for example, in a bean) in the request object. The servlet that generated the context generates a response by passing the request object to a JSP file that contains the HTML formatting. The <BEAN> tag provides access to the business logic.

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().

Migrating JSP 0.91 files to the Application Server Version 3
There are two options for migration. Perform *one* of the following migrations:

Migrate to JSP 1.0: It is recommended that you migrate JSPs developed under the Application Server Version 2 and develop new JSPs to conform to the JSP 1.0 Specification. Refer to the Sun JSP 1.0 Specification for details.

Migrate servlets or JSPs that use HttpServiceRequest and HttpServiceResponse:
If your servlets or JSPs developed under the Application Server Version 2 cast to methods of com.sun.server.http.HttpServiceRequest or com.sun.server.http.HttpServiceResponse, you must perform *one* of the following migration steps:

- Migrate HttpServiceRequest.setAttribute() to HttpServletRequest.setAttribute() and migrate HttpServiceResponse.callPage() to RequestDispatcher.
- Recompile your JSPs developed under the Application Server Version 2 before you use them with the Application Server Version 3. Recompiling is necessary because HttpServiceRequest and HttpServiceResponse are provided as interfaces (instead of classes) that are implemented by the Version 3 servlet engine.
  Note: If you do not recompile your servlets or JSPs, the Java Virtual Machine (JVM) will crash on Windows NT systems due to a bug in the JDK.

Please refer to Sun's JSP specification for more details:

http://java.sun.com/products/jsp/download.html

RELATED CONCEPTS

JSP and servlets
JavaServer Pages

RELATED TASKS

Migrating between JSP 0.91 and JSP 1.0 support

RELATED REFERENCES

JSP 1.0 programming reference

JSP 1.0 programming reference

IBM extensions and additions to JSP 1.0 syntax

The following sections describe the IBM extensions and additions to JSP 1.0 provided in the WebSphere Application Server Version 3. Refer to the Sun JSP 1.0 Specification for details about the base APIs.

JSP 1.0 syntax for variable data

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

The JSP tags for variable data are:

- <tsx:getProperty> syntax for getting the value of a bean to display in a JSP
- <tsx:repeat> syntax for repeating a block of HTML tagging that contains the <tsx:getProperty> syntax and the HTML tags for formatting content

<tsx:getProperty>

The <tsx:getProperty> is an IBM extension of the Sun JSP 1.0 <jsp:getProperty> tag. The IBM extension 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>.

The <tsx:getProperty> syntax is:

```
<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 of the attribute is case-sensitive and is the locale-independent name of the property.

Some examples are:

```
<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 the property 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 indexed properties. The optional index can be a constant (such as 2) or an index like the one described in the <tsx:repeat> section. 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 />
```

<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 ArrayIndexOutofBoundsException 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.

The <tsx:repeat> syntax is:

```
<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 default is 0.
- **end** An optional ending index value for this repeat block. The maximum value is 2,147,483,647. If the value of the end attribute is less than the value of the start attribute, the end attribute is ignored.

The 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	JSP file

`// 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>`
 	
Notes:
• 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.
	`<tsx:dbconnect id="conn"`
	`userid="alice"passwd="test"`
	`url="jdbc:db2:sample"`
	`driver="COM.ibm.db2.jdbc.app.DB2Driver"`
	`</tsx:dbconnect >`
	`<tsx:dbquery id="dynamic" connection`
	`="conn" >`
	`Select * from myTable;`
	`</tsx:dbquery>`
	`<tsx:repeat index=abc>`
	`<tsx:getProperty name="dynamic"`
	`property="col1(abc)" />`
	`</tsx:repeat>`
	
	Notes:
	• 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>. The examples produce the same output if all indexed properties have 300 or fewer elements. If there are more than 300 elements, Examples 1 and 2 will display all elements, while Example 3 will show only the first 300 elements.

Example 1 shows implicit indexing with the default start and default end index. 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 starting index. Although the index attribute is specified, the indexed property city 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 separately indexed. This capability is useful for interleaving properties on two beans, or properties that have sub-properties. In the example, two <tsx:repeat> blocks are nested to display the list of songs on each compact disc 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>
```

JSP syntax for database access

The Application Server Version 3 extends JSP 1.0 support by providing syntax for database access. The syntax makes 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 scope of all of the syntax is the JSP file in which they are embedded. Therefore, identifiers and other tag data can be accessed only within the page.

<tsx:dbconnect>

Use the <tsx:dbconnect> syntax to specify information needed to make a connection to a JDBC or an ODBC database. The <tsx:dbconnect> syntax does not establish the connection. Instead, the <tsx:dbquery> and <tsx:dbmodify> syntax are used to reference a <tsx:dbconnect> in the same JSP file and establish the connection.

The <tsx:dbconnect> syntax is:

```
<tsx:dbconnect id="connection_id"
        userid="db_user" passwd="user_password"
        url="jdbc:subprotocol:database"
        driver="database_driver_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 hardcoding 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 hardcoding 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. 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"
  ```
  Note: 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.
  All of the elements shown in the example XML file need to be specified. However, an empty element (such as <url></url>) is valid.

When the JSP file is compiled into a servlet, the Java processor adds the Java coding for the <tsx:dbconnect> syntax to the servlet's service() method, which means a new database connection is created for each request for the JSP.

<tsx:userid> and <tsx:passwd>
Instead of hardcoding the user ID and password in the <tsx:dbconnect>, you can use the <tsx:userid> and <tsx:passwd> to accept user input for the values and then add that data to the request object where it can be accessed by a JSP (such as the JSP10Employee.jsp example) that requests the database connection.

The <tsx:userid> and <tsx:passwd> must be used within a <tsx:dbconnect> tag. The <tsx:userid> and <tsx:passwd> syntax is:

```
<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.

<tsx:dbquery>
Use the <tsx:dbquery> syntax to establish a connection to a database, submit database queries, and return the results set.

The <tsx:dbquery>:

- 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>.
- Establishes a new connection
- Retrieves and caches data in the results object
- Closes the connection (releases the connection resource)

The <tsx:dbquery> syntax is:

```
<%— 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 and the effective limit is determined by the number of records and the system caching capability.
- **SELECT command** and **JSP syntax** Because the <tsx:dbquery> 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.

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 that 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>
```

<tsx:dbmodify>

Use the <tsx:dbmodify> syntax to establish a connection to a database and then add records to a database table.

The <tsx:dbmodify>:

- 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>.
- Establishes a new connection.
- Updates a table in the database.
- Closes the connection (releases the connection resource).

The <tsx:dbmodify> syntax is:

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

where:

- **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.
- **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 <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>
```

Displaying query results

To display the query results, use the <tsx:repeat> and <tsx:getProperty> syntax. 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> property attribute) to retrieve the value. An example is:

```
<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>
```

Please refer to Sun's JSP specification for more details:

http://java.sun.com/products/jsp/download.html

RELATED CONCEPTS

JSP and servlets
JavaServer Pages

RELATED TASKS

Migrating between JSP 0.91 and JSP 1.0 support

RELATED REFERENCES

JSP 0.91 programming reference

Generated servlet file names

The generated servlet code resides under the pagecompile package in the JSP Page Compile Generated Code project. The file names of the generated servlets differ according to the state of the JSP Execution Monitor. The following chart summarizes the various scenarios.

	JSP Execution Monitor enabled	**JSP Execution Monitor disabled**
JSP source debugging feature enabled	_<JSP file name>_xjsp_debug_jspsrc_ <random number>.java	_<JSP file name>_xjsp_jspsrc_<random number>.java
Load generated servlet externally feature disabled	_<JSP file name>_xjsp_debug.java	_<JSP file name>_xjsp.java
Load generated servlet externally feature enabled	_<JSP file name>_xjsp_debug_<random number>.java	_<JSP file name>_xjsp_<random number>.java

where

- <JSP file name> is the JSP file name without the path and the '.jsp' extension.
- <random number> is a randomly generated number.

RELATED CONCEPTS

JSP Execution Monitor

RELATED TASKS

Debugging JSP-generated servlet code
Using the JSP Execution Monitor
Enabling JSP source debugging
Loading generated servlet externally

RELATED REFERENCES

JSP debug flow

Notices

Note to U.S. Government Users Restricted Rights — Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.
IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:
IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.
This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Lab Director
IBM Canada Ltd.
1150 Eglinton Avenue East
Toronto, Ontario M3C 1H7
Canada

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.
The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:
© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. 1997, 2000. All rights reserved.

Programming interface information

Programming interface information is intended to help you create application software using this program.

General-use programming interfaces allow the customer to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks and service marks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

- AIX
- AS/400
- DB2
- CICS
- CICS/ESA
- IBM
- IMS
- Language Environment
- MQSeries
- Network Station
- OS/2
- OS/390
- OS/400
- RS/6000
- S/390
- VisualAge
- VTAM
- WebSphere

Lotus, Lotus Notes and Domino are trademarks or registered trademarks of Lotus Development Corporation in the United States, or other countries, or both.

Tivoli Enterprise Console and Tivoli Module Designer are trademarks of Tivoli Systems Inc. in the United States, or other countries, or both.

Encina and DCE Encina Lightweight Client are trademarks of Transarc Corporation in the United States, or other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

ActiveX, Microsoft, SourceSafe, Visual C++, Visual SourceSafe, Windows, Windows NT, Win32, Win32s and the Windows logo are trademarks or registered trademarks of Microsoft Corporation in the United States, or other countries, or both.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Intel and Pentium are trademarks of Intel Corporation in the United States, or other countries, or both.

Other company, product, and service names, which may be denoted by a double asterisk(**), may be trademarks or service marks of others.