# INTRODUCTION TO IMPLICIT OBJECTS

Implicit objects are created automatically when a Web server processes a JSP page. The available implicit objects include `application`, `config`, `exception`, `out`, `page`, `pageContext`, `request`, `response` and `session`. Each object is used to perform a specific task, such as handling errors, sending text generated by a JSP page to a Web browser or interpreting information submitted by a form on a Web page.

Implicit objects are available for use in every JSP page you create. You do not have to write code that imports or instantiates an implicit object.

### Object Scope

The scope of an object determines where the object can be accessed in an application. For example, the `session` object has session scope, which means that the object can be accessed by any JSP page processed during a session. Most implicit objects have page scope. When an object has page scope, the object can be accessed only in the JSP page in which the object was created. You can access implicit objects only from within scriptlets or expressions on a JSP page. Implicit objects are not available for use in directives, such as the `page` directive.

| OBJECT: | SCOPE: |
|---|---|
| application | application |
| config | page |
| exception | page |
| out | page |
| page | page |
| pageContext | page |
| request | request |
| response | page |
| session | session |

### Class Files

Since JSP pages use the underlying servlet technology of the Web server, implicit objects are usually derived from class files that are part of the servlet packages. For more information about implicit objects and the servlet packages, you can consult the Java SDK documentation.

| OBJECT: | CLASS: |
|---|---|
| application | javax.servlet.ServletContext |
| config | javax.servlet.ServletConfig |
| exception | java.lang.Throwable |
| out | javax.servlet.jsp.JspWriter |
| page | java.lang.Object |
| pageContext | javax.servlet.jsp.PageContext |
| request | javax.servlet.ServletRequest |
| response | javax.servlet.ServletResponse |
| session | javax.servlet.http.HttpSession |

## IMPLICIT OBJECTS

**`application`**

The `application` object is used to store information about an application. An application is a collection of JSP pages stored in a specific directory and its subdirectories on a Web server.

**`pageContext`**

The `pageContext` object is used to access the characteristics of a JSP page that are specific to the Web server processing the page.

**`config`**

The `config` object is used to store information about the configuration of the environment in which a JSP page is processed on a Web server.

**`request`**

The `request` object is used to store information supplied by a client, such as data submitted in a form or the IP number and name of the client computer.

**`exception`**

The `exception` object is used to handle errors that may occur when a JSP page is processed. The `exception` object also stores error information.

**`response`**

The `response` object is used to store information generated by a Web server before the information is sent to a client.

**`out`**

The `out` object is used to send output generated by a JSP page to a client's Web browser.

**`page`**

The `page` object is used to store information about a JSP page while the page is being processed. The `page` object is not typically accessed from within a JSP page.

**`session`**

The `session` object is used to store information associated with a session. A sessions starts when a client requests a JSP page from a Web site and ends when the client does not request another page for a specific period of time or the session is abandoned.

# CREATE A FORM

Adding a form to a Web page allows you to gather data from users who visit the page. A form can be placed anywhere between the `<body>` and `</body>` tags in an HTML document. The body of your Web page can include as many forms as you need.

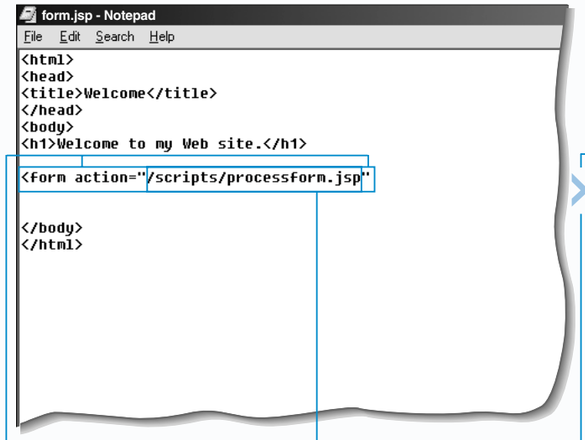
You use the `<form>` tag to create a form and the `action` attribute to specify the location and name of the JSP page that will process the data entered into the form. If the JSP page is stored in the same directory as the Web page containing the form, you only have to specify the name of the JSP page. If the JSP page is not stored on the same Web server as the Web page containing the form, you must specify the full URL of the JSP page.

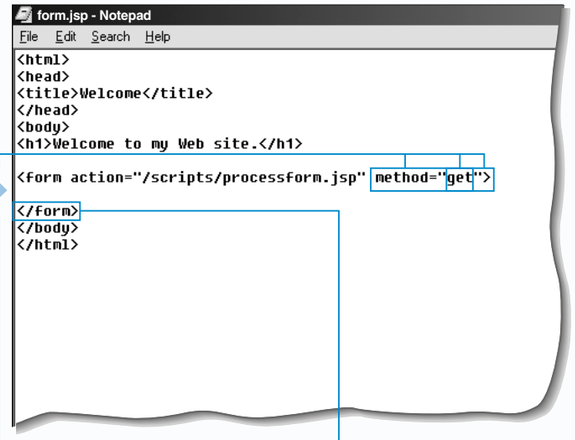
You must also specify which method the form will use to pass data to the JSP page. There are two methods the form can use–`get` and `post`. The method you should use depends on the amount of data that will be passed. The `get` method sends data to the JSP page by appending the data to the URL of the page. The `post` method sends the data and the URL separately. The `get` method is faster than the `post` method and is suitable for small forms. The `post` method is suitable for large forms that will send more than 2000 characters to the JSP page.

Unlike other technologies used to process form information, JavaServer Pages can automatically determine whether a form is submitting data using the `get` or `post` method and then retrieve the information.

For information about creating a JSP page that processes data from a form, see page 84.

## CREATE A FORM




**1** Type **<form action="">** where you want to add a form to a Web page.

**2** Between the quotation marks, type the location and name of the JSP page that will process the data entered into the form.

**3** Type **method="">**.

**4** Between the quotation marks, type the method the form will use to pass data to the JSP page.

**5** Type **</form>** where you want to end the form.

■ You can now add elements to the form.





# ADD ELEMENTS TO A FORM

Elements are areas in a form where users can enter data and select options. The most commonly used element is a text box, which allows users to enter a single line of data into a form. Text boxes are often used for entering names, addresses and other short responses.

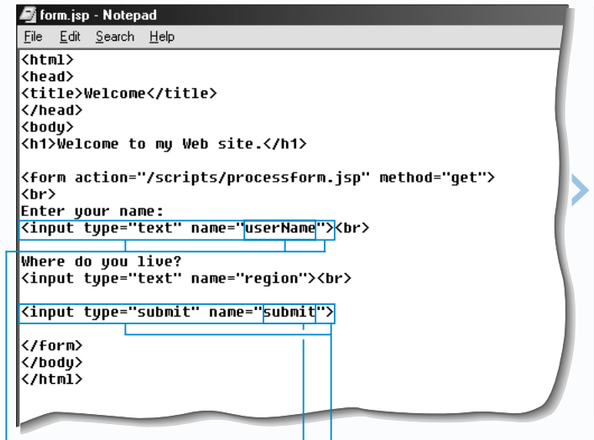
Elements you add to a form must be placed between the `<form>` and `</form>` tags. A form can contain as many elements as you need.

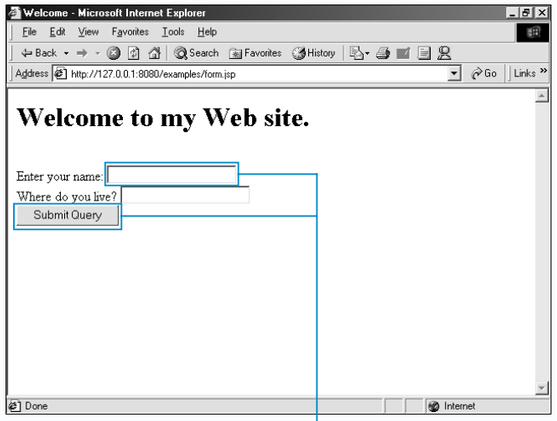
There are many different types of elements you can add to a form, such as text areas and check boxes. Text areas allow users to enter several lines or paragraphs of text, while check boxes let users select options on a form. For information about commonly used elements, see page 82.

Each form element has attributes, such as `name`, `type` and `size`, which offer options for the element. The `name` attribute allows you to provide a name for an element. The name you specify is used by the JSP page that processes the form to identify the element and access the information in the element. A name can contain letters and numbers, but should not contain spaces or punctuation. If you want to include spaces in a name, use an underscore character (_) instead.

You must add a submit button to every form you create. The submit button allows users to send the data they entered into the form to the Web server. When the Web server receives data from a form, the server transfers the data to the JSP page that will process the data. The JSP page can then perform an action with the data, such as storing the data in a database or displaying the information in a Web browser.

## ADD ELEMENTS TO A FORM




**1** To add a text box to a form, type **<input type="text" name="">** between the `<form>` and `</form>` tags.

**2** Between the quotation marks, type a word that describes the text box.

**3** To add a submit button to the form, type **<input type="submit" name="">**.

**4** Between the quotation marks, type a word that describes the button.

**5** Display the Web page in a Web browser.

■ The Web browser displays the text box and submit button.

# FORM ELEMENTS

An element is an area in a form where users can enter data or select options. There are several different types of elements you can add to a form.

Most elements require you to specify attributes that determine how the element will appear on a Web page.

You can find more information about form elements and attributes at the www.w3.org/TR/1999/REC-html401-19991224/interact/forms Web site.

## COMMONLY USED ATTRIBUTES

### Type

The `type` attribute allows you to specify the kind of element you want to use.

### Name

The `name` attribute allows you to specify a name for an element. The JSP page that will process data from the element uses the `name` attribute to identify the data. Element names can contain more than one word, but should not contain spaces or special characters.

### Value

The `value` attribute allows you to specify a value for an element. If an element displays a button, you can use the `value` attribute to specify the text that will appear on the button.

### Maxlength

The `maxlength` attribute allows you to restrict the number of characters a user can enter into an element.

### Size

The `size` attribute allows you to specify the width of an element.

### Checked

The `checked` attribute allows an element to display a selected option by default.

## COMMONLY USED ELEMENTS

### Password Box

A password box allows users to enter private data. When a user types data into a password box, an asterisk (*) appears for each character, which prevents others from viewing the data on the screen. A password box does not protect the data from being accessed as it is transferred over the Internet. You must set the `type` attribute to `password` and use the `name` attribute to create a password box. You may also want to use the `value`, `maxlength` and `size` attributes.

```
Password Please <input type="password"
name="secretWord" value="password" maxlength="20">
```

Password Please ********

### Drop-Down List

The `select` element displays a drop-down list that allows users to select an option from a list of several options. For example, a drop-down list can be used to allow users to select one of three shipping methods. You must use the `name` attribute to create a drop-down list. You use the `<option>` tag with the `value` attribute to add options to the list.

```
How would you like your products shipped?
<select name="shipMethod">
<option value="air">Air</option>
<option value="land">Land</option>
<option value="sea">Sea</option>
</select>
```

How would you like your products shipped? Air
Air
Land
Sea





## COMMONLY USED ELEMENTS

### Text Box

A text box allows users to enter a single line of text, such as a name or telephone number. You must set the `type` attribute to `text` and use the `name` attribute to create a text box. You may also want to use the `maxlength` and `size` attributes.

```
First Name <input type="text" name="firstName" maxlength="20">
```

First Name

### Text Area

The `textarea` element displays a large text area that allows users to enter several lines or paragraphs of text. A large text area is ideal for gathering comments or questions from users. You must use the `name` attribute to create a text area.

```
Questions? <textarea name="userQuestions"></textarea>
```

Questions?

### Check Box

Check boxes allow users to select one or more options. For example, check boxes can be used to allow users to specify which states they have visited. You must set the `type` attribute to `checkbox` and use the `name` and `value` attributes to create a check box. You may also want to use the `checked` attribute.

```
Which states have you visited in the past year?<br>
New York <input type="checkbox" name="states" value="New York" checked>
California <input type="checkbox" name="states" value="California">
Texas <input type="checkbox" name="states" value="Texas">
```

Which states have you visited in the past year?
New York ☑ California ☐ Texas ☐

### Radio Button

Radio buttons allow users to select only one of several options. For example, radio buttons can be used to allow users to specify if they are male or female. You must set the `type` attribute to `radio` and use the `name` and `value` attributes to create a radio button. You may also want to use the `checked` attribute.

```
What is your gender?<br>
Female <input type="radio" name="gender" value="female" checked>
Male <input type="radio" name="gender" value="male">
```

What is your gender?
Female ◉ Male ○

### Submit Button

A submit button allows users to send data in the form to the JSP page that will process the data. You must add a submit button to each form you create. You must set the `type` attribute to `submit` to create a submit button. You may also want to use the `name` and `value` attributes.

```
<input type="submit" name="submit" value="Submit Now">
```

Submit Now

### Reset Button

A reset button allows users to clear the data they entered into a form. A user cannot redisplay data that has been cleared. Reset buttons are commonly used in forms that have many text boxes. You must set the `type` attribute to `reset` to create a reset button. You may also want to use the `value` attribute.

```
<input type="reset" value="Click to Reset">
```

Click to Reset

# PROCESS DATA FROM A FORM
## Using the getParameter Method

After creating a form on a Web page, you can create a JSP page that will process data submitted in the form. The `getParameter` method of the `request` object allows a JSP page to access form data.

You must specify the name of the form element you want to access using the `getParameter` method. The name you specify must be exactly the same as the name that was assigned to the element when it was created. If the element name you specify does not exist in the form, the `getParameter` method will return a null value.

Once a JSP page has accessed data from a form element, the page can perform a task, such as storing the data in a file or a database. While JSP pages that process data from forms do not need to generate any output, these pages typically produce an acknowledgement message or redirect a client to another page.

Some Web servers require JSP pages that process data from a form to be saved in a specific directory. You should check the latest documentation for your Web server to determine where you should save a JSP page that processes form information. If your Web server does not require the JSP page to be saved in a specific directory, you may want to save the page in the same directory as the form it processes.

After saving the JSP page, you should review the code for the Web page that contains the form to verify that the `action` attribute displays the correct filename and location for the JSP page.

Although the `getParameter` method is still commonly used, the method is deprecated. This means that the `getParameter` method is no longer recommended and will eventually become obsolete. The `getParameterValues` method is now the preferred method for accessing information in a form element. For more information about the `getParameterValues` method, see page 86.

### Apply It

When you use the `getParameter` method of the `request` object to access data from a form, the data is retrieved as a string value that can be assigned to a variable and then used in your code.

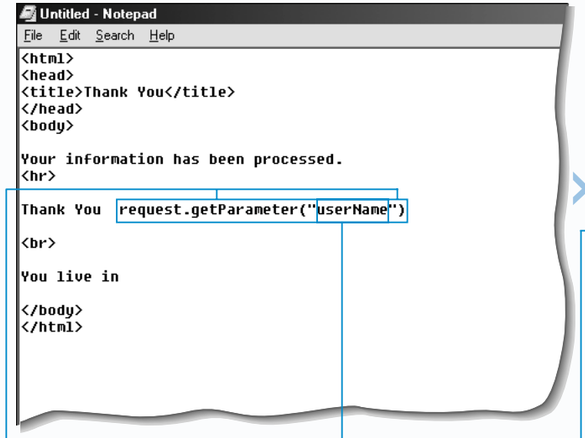
**TYPE THIS:**

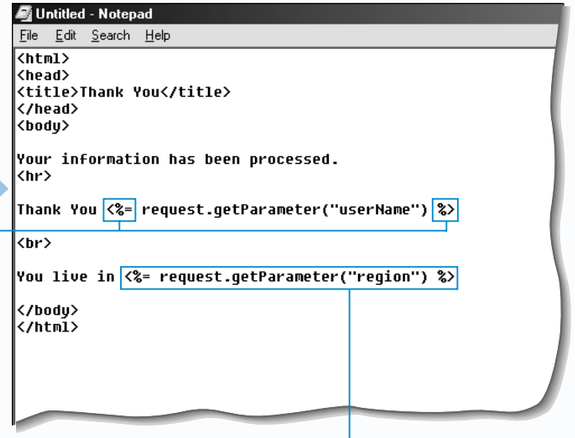
```
<%
String id = request.getParameter("userName");
String locale = request.getParameter("region");
String message = "Login Name:" + id + "<br>";
message = message + "Location:" + locale;
%>
<html>
<head><title>Thank You</title></head>
<body>
Your information has been processed.<hr>
<%= message %>
</body>
</html>
```

**RESULT:**

Your information has been processed.
---
Login Name:Barry
Location:Texas

### USING THE GETPARAMETER METHOD



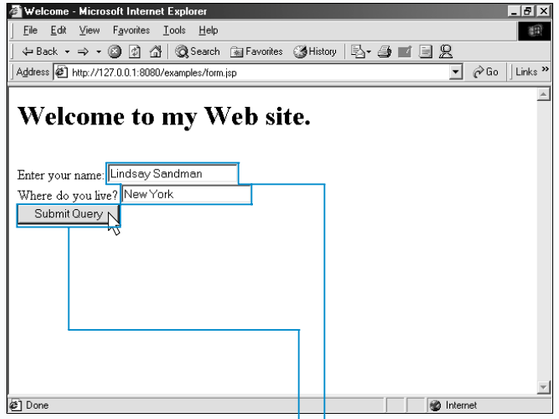

**1** In the JSP page you want to process data from a form, type **request.getParameter("")**.

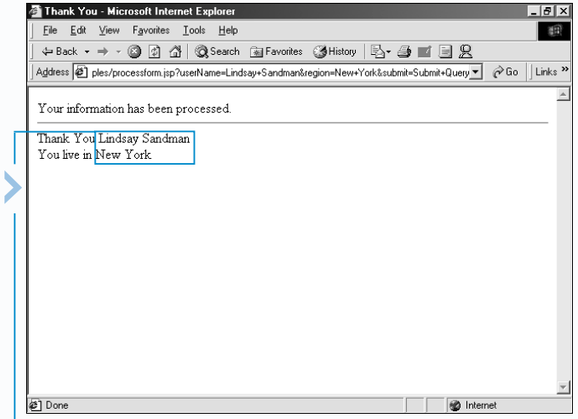
**2** Between the quotation marks, type the name of the form element you want to access.

**3** Type the code that uses the data from the form element.

**4** Repeat steps 1 to 3 for each form element you want to process.

**5** Save the page with the .jsp extension.




**PROCESS FORM INFORMATION**

**1** In a Web browser, display the Web page containing the form you want to process.

**2** Enter data into the form.

**3** Click the submit button to pass the data in the form to the JSP page.

■ The Web browser displays the result of using the `getParameter` method to process data from the form.

# PROCESS DATA FROM A FORM
## Using the getParameterValues Method

The `getParameterValues` method of the `request` object can be used to access the data passed by a form. The `getParameterValues` method is the preferred method for accessing form data, although the `getParameter` method can also be used. For information about the `getParameter` method, see page 84.

The `getParameterValues` method is particularly useful for accessing a form element that can contain multiple values. For example, some drop-down lists allow users to select more than one option. The `getParameterValues` method returns the data in a form element as an array of string values.

You must specify the name of the form element you want to access using the `getParameterValues` method. The name you specify must be exactly the same as the name that was assigned to the element when it was created. If the element name you specify does not exist in the form, the `getParameterValues` method will return a null value.

You can assign the data returned by the `getParameterValues` method to an array variable. This allows you to work with the data in the form element. For example, you can use a `for` loop to display each value stored in the element.

When saving a JSP page that processes data from a form, you should check the latest documentation for your Web server to determine where you should save the page. Some Web servers require you to save JSP pages that process form data in a specific directory. If your Web server does not specify the directory you should use, you may want to save the JSP page in the same directory as the form it processes.

After saving the JSP page, you should review the code for the Web page that contains the form to verify that the `action` attribute displays the correct filename and location for the JSP page.

### Apply It

When processing data from a form, you should include code in your JSP page that checks the validity of data a user submits in the form. For example, if you want users to select at least two options from a drop-down list, you can add error-checking code that ensures two selections were made.

**FORM:**

Please select at least two catergories you would like more information about:

Products
Order Information
Contact Information
Shipping Information
Employment

Submit

**IN THE JSP PAGE, TYPE:**

```
<%
String[] names = request.getParameterValues("info");
if (names.length > 1)
{
      for (int x = 0; x < names.length; x++)
            out.print(names[x] + "<br>");
}
else
{
      out.print("Please select at least 2 items");
}
%>
```

**RESULT:**

Please select at least 2 items.

### USING THE GETPARAMETERVALUES METHOD

Untitled - Notepad

File Edit Search Help

```
<html>
<head>
<title>Thank You</title>
</head>
<body>

Your information has been processed. You Selected:<br>

<%
request.getParameterValues("area");

%>

</body>
</html>
```

Untitled - Notepad

File Edit Search Help

```
<html>
<head>
<title>Thank You</title>
</head>
<body>

Your information has been processed. You Selected:<br>

<%
String[] names = request.getParameterValues("area");

for (int x = 0 ; x < names.length ; x++)
     out.print(names[x] + "<br>");
%>

</body>
</html>
```

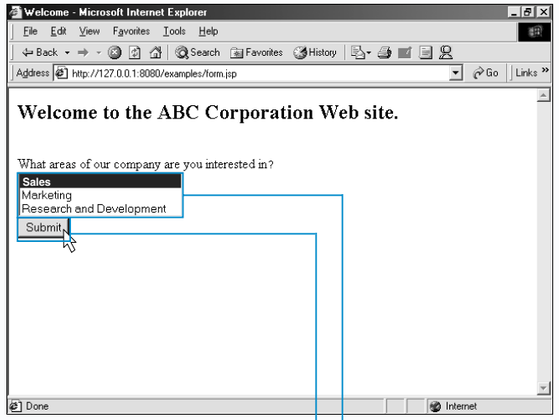
1 In the JSP page you want to process data from a form, type **request.getParameterValues("")**.

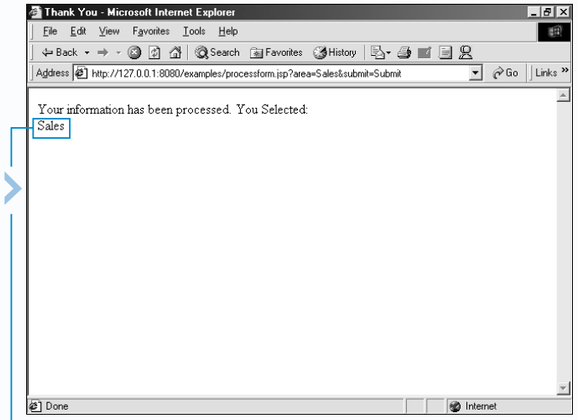
2 Between the quotation marks, type the name of the form element you want to access.

3 Type the code that assigns the data from the form element to an array variable.

4 Type the code that uses the data from the form element.

5 Save the page with the .jsp extension.




**PROCESS FORM INFORMATION**

1 In a Web browser, display the Web page containing the form you want to process.

2 Enter data into the form.

3 Click the submit button to pass the data in the form to the JSP page.

4 The Web browser displays the result of using the `getParameterValues` method to process data from the form.

# DETERMINE THE ELEMENTS IN A FORM

The `getParameterNames` method of the `request` object can be used to retrieve the name of each element a form contains. You may want to determine the names of elements in a form to verify that the form contains the correct elements. Being able to determine the names of form elements also allows you to create a single JSP page that can process data from several different forms.

Form elements do not have to contain data in order to be included in the list retrieved by the `getParameterNames` method. If you gave the submit button on your form a name, the name will be included in the list of element names.

An efficient way to work with the element names retrieved by the `getParameterNames` method is to cast the names as a collection, or iteration, that can be used by the `Iterator` interface. An interface is a set of method declarations that offers the same functionality as a class.

You must use the `page` directive with the `import` attribute to import the `Iterator` interface from the `java.util` package. For more information about the `page` directive, see page 74. For more information about the `Iterator` interface and the `java.util` package, you can refer to the Java SDK documentation.

A `for` statement can be used to create a loop that will process each element name in the collection. The `next` method of the `Iterator` interface controls the loop, so a re-initialization expression is not required in the `for` statement.

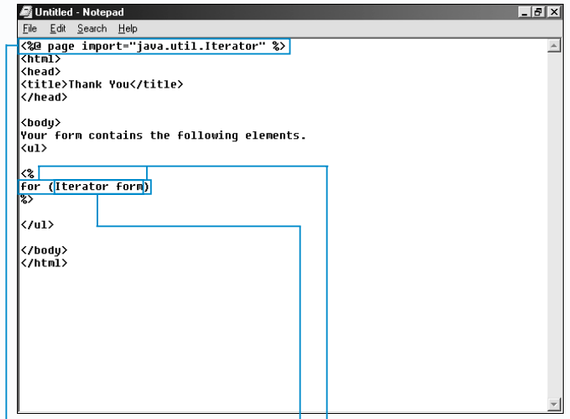
When you create the code for the form whose elements you want to determine, the `action` attribute of the `<form>` tag must specify the name and location of the JSP page you set up to process form information.

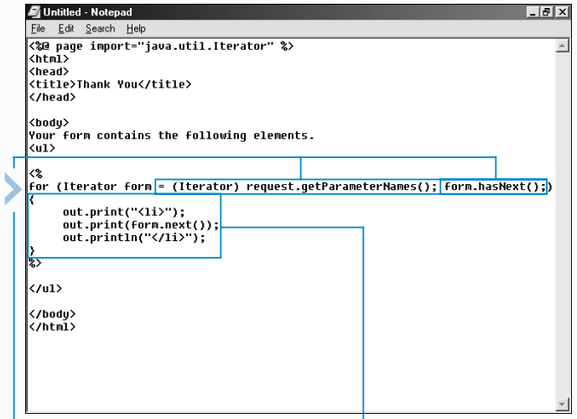
## Extra

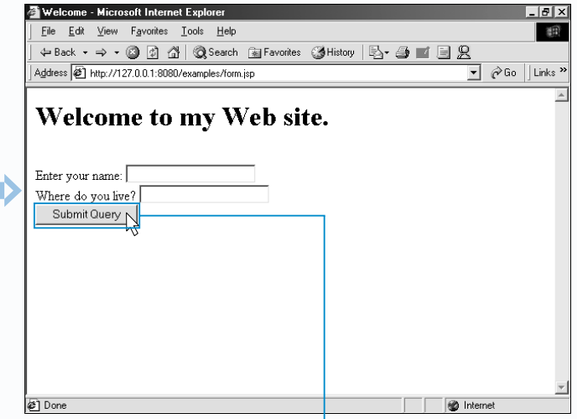
The `Iterator` interface includes three methods that can be used to work with a collection of elements.

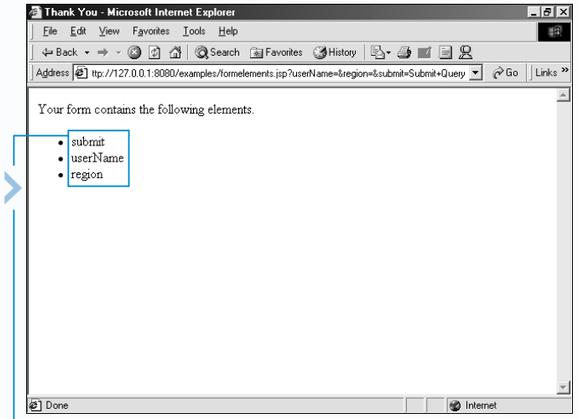
| NAME: | DESCRIPTION: |
|---|---|
| `hasNext` | This method is used to determine if there are any elements left to process in the collection. The `hasNext` method returns a boolean value of true if the collection has another element that can be accessed. If there are no more elements in the collection, a value of false is returned. |
| `next` | This method returns the next element in the collection. If there are no more elements in the collection, an error is generated. |
| `remove` | This method discards the last element returned by the `next` method. If the `next` method is not used before the `remove` method, an error is generated. |

## DETERMINE THE ELEMENTS IN A FORM




**1** In the first line of code in the JSP page, type **<%@ page import="java.util.Iterator" %>** to import the `Iterator` interface from the `java.util` package.

**2** To create a loop that will process each element name in a form, type **for ()**.

**3** To create the initialization expression for the `for` statement, type **Iterator** followed by a name for the element names in a form.

**4** Type **=** followed by **(Iterator) request.getParameterNames();** to cast the element names retrieved by the `getParameterNames` method as a collection.

**5** To create a condition for the `for` statement, type the name of the collection followed by **.hasNext();**.

**6** Type the code that will process each element in the collection. Enclose the code in braces.

**7** Save the page with the .jsp extension.

**8** In a Web browser, display the form whose elements you want to determine.

**9** Click the submit button to pass the form information to the JSP page.

**10** The Web browser displays the name of each element in the form.

# ACCESS CLIENT INFORMATION

A JSP page can access information about a client computer, such as the IP address and name of the computer. Accessing information about a client computer is useful if you want to verify the identity of a client or perform an administrative task, such as creating a log that documents Web site usage.

Every computer connected to a network using the TCP/IP protocol has a unique IP address. The `getRemoteAddr` method of the `request` object is used to access a client computer's IP address and return the IP address as a string value.

The `getRemoteHost` method of the `request` object allows a JSP page to access the name of a client computer. This method returns a string value containing the full domain name of the client, such as computer2.abccorp.com. The `getRemoteHost` method retrieves the name of a client from your Web server, which uses a Domain Name System (DNS) server to determine the name based on the client computer's IP address. This means that your Web server must be able to communicate with a DNS server before the `getRemoteHost` method can access the name of a client. If the method cannot access the name of a client, it will return the client's IP address.

A JSP page can use the `getServerPort` method of the `request` object to access the port number a client is using for a request. This method returns an integer that indicates which server port received the request. Using the `getServerPort` method is useful when your server uses different ports for different types of programs. For example, if administrative programs use a specific port on your server, accessing the port number lets you determine whether a client is an administrator or a regular user. This allows you to customize the content of a JSP page depending on the type of client accessing the page.

## Apply It

Once you have accessed the IP address of a client computer, you can use this information to grant or deny the client access to your JSP page. You can use the `indexOf` method to compare an IP address you specify to a client computer's IP address. In the following example, a welcome message appears when a client with an IP address beginning with 127.0.0 accesses the JSP page.

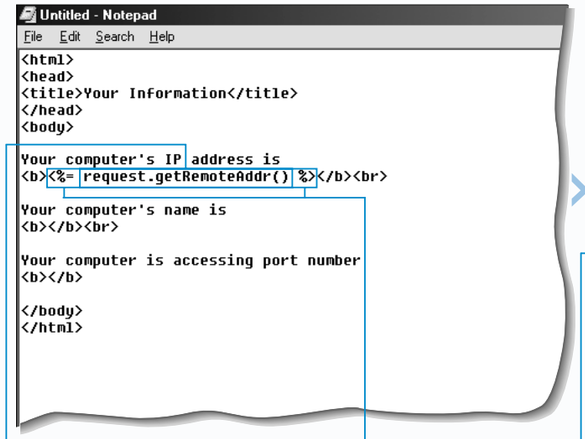
TYPE THIS:

```
<%
String ipNumber = request.getRemoteAddr();
int postion = ipNumber.indexOf("127.0.0");

if (postion == 0)
      {
            out.print("Welcome to my Web site");
      }
      else
      {
            out.print("You are not authorized to continue");
      }
%>
```

RESULT:

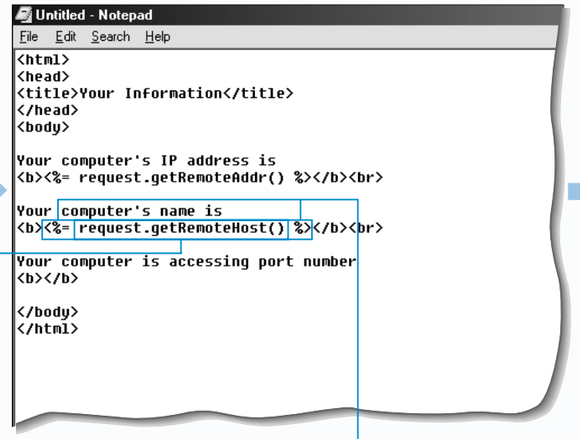
Welcome to my Web site

## ACCESS CLIENT INFORMATION



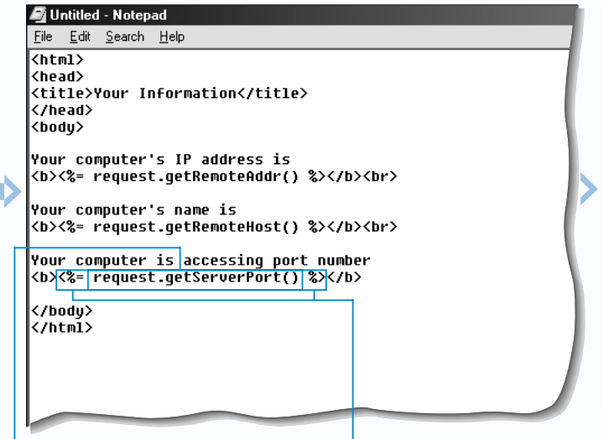

1 To access the IP address of a client computer, type **request.getRemoteAddr( )**.

2 Type the code that uses the IP address.



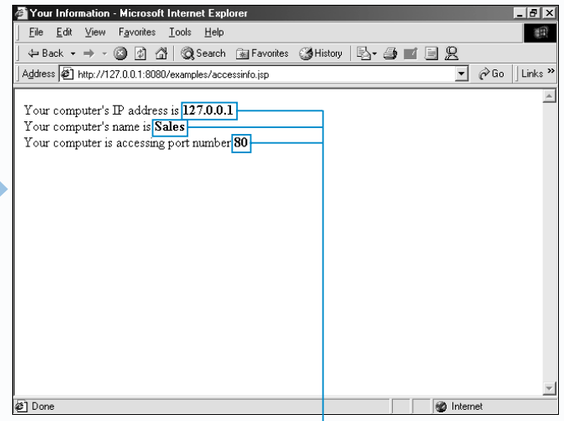

3 To access the name of a client computer, type **request.getRemoteHost( )**.

4 Type the code that uses the name.




5 To access the port number for a client request, type **request.getServerPort( )**.

6 Type the code that uses the port number.




7 Save the page with the .jsp extension and then display the JSP page in a Web browser.

■ The Web browser displays the results of accessing client information.

# WORK WITH THE BUFFER

The buffer is a section of the Web server's memory where a JSP page can be stored temporarily. When a JSP page is being processed, the data for the page is stored in the buffer instead of being sent directly to a user's Web browser. When the buffer is full or the entire JSP page has been generated, the Web server automatically sends the contents of the buffer to the Web browser.

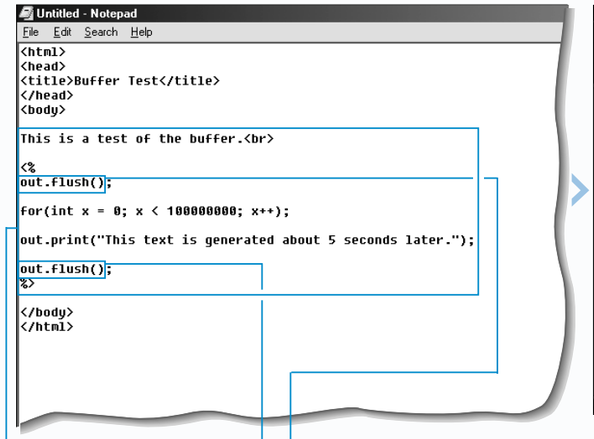
The `flush` method of the `out` object forces the Web server to send the contents of the buffer to the Web browser. This allows you to control when a user will see information from your JSP page. For example, if your JSP page displays a banner image followed by a large amount of data from a database, you can use the `flush` method to force the JSP page to display the banner first.

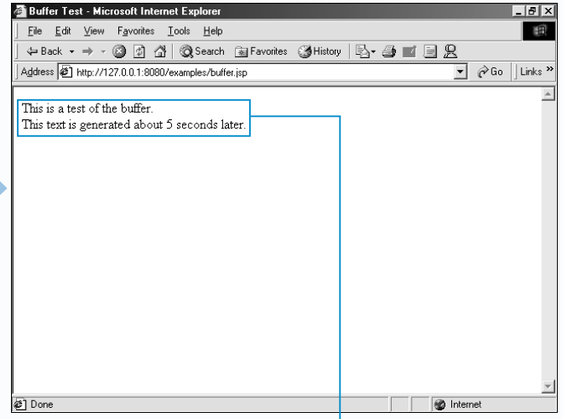
When you use the `flush` method, all the information in the buffer is immediately sent to the user's browser and the buffer is emptied. The next time the `flush` method is called, the contents of the buffer will include only the information processed since the `flush` method was last used.

You can use the `clearBuffer` method of the `out` object to clear information from the buffer before the information is sent to a user's Web browser. The Web server deletes any information that was processed and added to the buffer since the `clearBuffer` method was last called or since the beginning of the JSP page.

Deleting the contents of the buffer is useful when an error occurs in a JSP page. For example, if there is information in the buffer and the JSP page detects an error, you can clear the information in the buffer and display an error message in the user's Web browser.

## SEND CONTENTS OF BUFFER TO WEB BROWSER




1 Type the code you want to execute to display information in a user's Web browser.

2 Type **out.flush()** directly below the information you want to send to a user's Web browser.

3 Repeat step 2 for each section of code you want to send to a user's Web browser at a time.

4 Save the page with the .jsp extension and then display the JSP page in a Web browser.

■ The result of sending the contents of the buffer to the Web browser is displayed.



**Apply it**

The size allocated for the buffer on the Web server depends on a number of parameters, such as the type of Web server you are using. On Windows platforms, the default size of the Tomcat Web server's buffer is 8 KB, or 8192 bytes. You can verify the size of the buffer on your Web server using the `getBufferSize` method of the `response` object.

TYPE THIS:

```
The current size of the buffer, in bytes, is:
<%= response.getBufferSize() %>
```

RESULT:

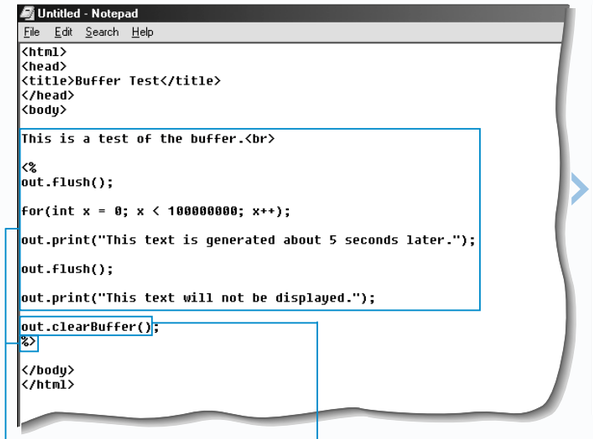
The current size of the buffer, in bytes, is: 8192

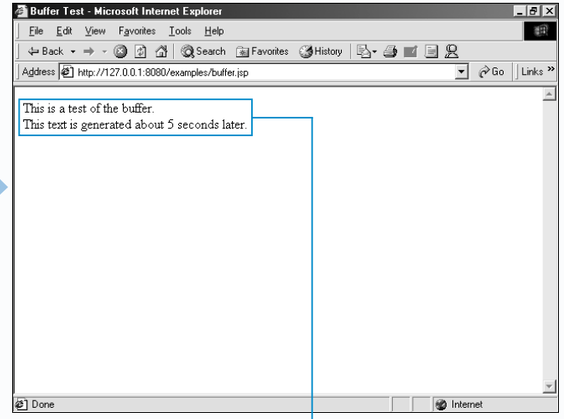
You can turn off buffering for specific JSP pages using the `page` directive. This is useful for JSP pages that require a small amount of processing. When the buffer is turned off, the Web server will send information to a user's Web browser as the information is generated from the JSP code. The `page` directive should be placed before any HTML code in a JSP page. You may not be able to turn off buffering for some Web servers, such as the Tomcat Web server.

TYPE THIS:

```
<%@ page buffer = "none" %>
```

## DELETE BUFFER CONTENTS




1 Type the code you want to execute to display information in a user's Web browser.

2 Type **out.clearBuffer()** where you want to delete the contents of the buffer.

3 Save the page with the .jsp extension and then display the JSP page in a Web browser.

■ The Web browser displays the information from the JSP page. Any information that was added to the buffer after the last `flush` method does not appear.

# ENCODE A URL

A session is started for each user who requests a JSP page from your Web site. When a session is created, a session ID is assigned to identify each user. By default, the session ID is stored on the user's computer using a cookie. Unfortunately, many users disable the Web browser's cookie features or use Web browsers that do not support cookie technology. Filtering software can also prevent the exchange of cookie information between clients and servers.

URL encoding, or rewriting, is the process of adding the session ID to a URL in a JSP page. This process allows the Web server to keep track of a client session when cookie technology is not supported. You use the `encodeURL` method of the `response` object to modify a URL in a page.

The `encodeURL` method first determines if the client supports the use of cookies. If the client does not support the use of cookies, the `encodeURL` method adds the session ID to the end of the URL that is passed to the method as an argument. If the encodeURL method determines that the client supports the use of cookies, the URL that is passed to the method is inserted into the HTML code without any modifications.

You should use the `encodeURL` method to generate any URL in the HTML code. If a client that does not support cookies accesses a URL that has not been rewritten, a new session will be created and the information from the previous session will be lost.

You can easily verify that URL encoding is being performed by viewing the URL of the Web page, which is typically displayed in the location or address box of the Web browser.

## Extra

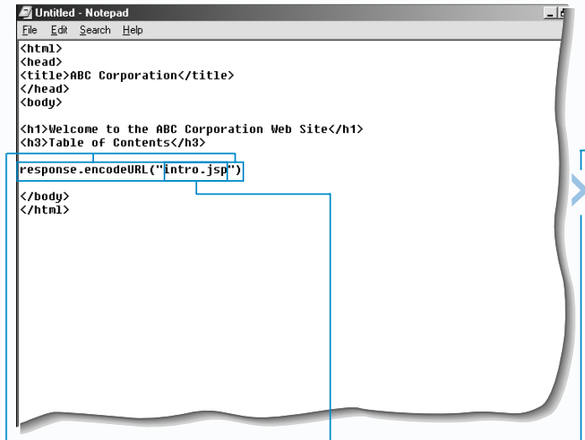
Cookies can be disabled in most browsers by modifying the Web browser security or file settings. Many Web sites offer reduced features and functionality if cookies are not supported by the client Web browser.

The `sendRedirect` method of the `response` object is used to redirect users to another Web page automatically. If you need to keep track of a client session when redirecting the user to another Web page, you should use the `sendRedirect` method in conjunction with the `encodeRedirectURL` method of the `response` object. The `encodeRedirectURL` method appends the session ID to the redirect URL when necessary, ensuring that session information is maintained even for users with browsers that do not support cookie technology.

**Example:**

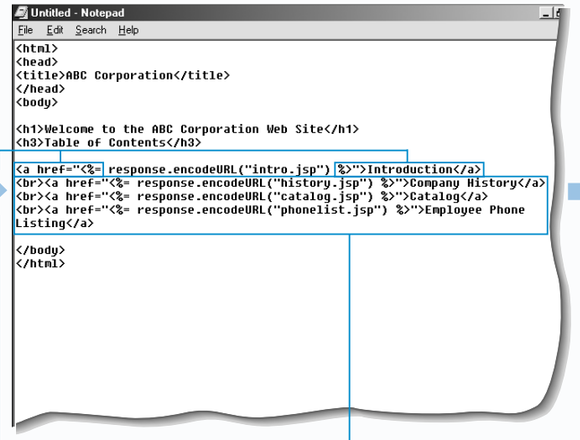
```
response.sendRedirect(response.encodeRedirectURL("errorPage.jsp"));
```

## ENCODE A URL



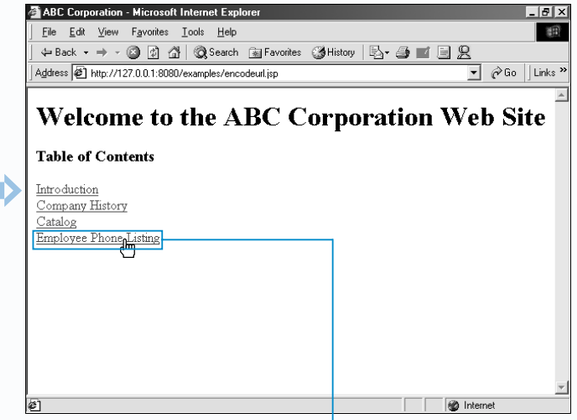

**1** To encode a URL, type **response.encodeURL("")**.

**2** Between the quotation marks, type the URL you want to be rewritten when cookie technology is not supported.




**3** Type the code that uses the encoded URL.

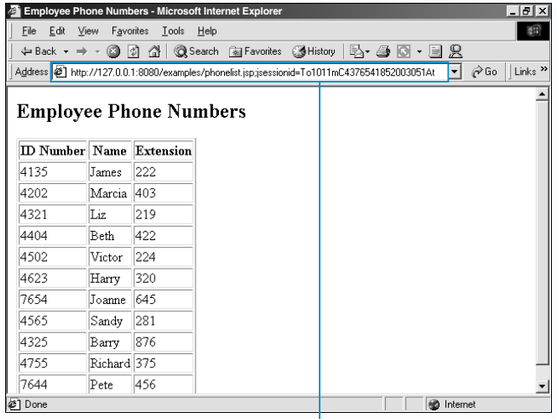
**4** Repeat steps 1 to 3 for each URL you want to encode in the HTML code.




**5** Save the page with the .jsp extension and then display the JSP page in a Web browser that does not support cookie technology.

■ The Web browser displays the page with the encoded URLs.

**6** Click a link to display another JSP page in the Web site.




■ The linked JSP page is displayed.

■ The location or address box displays the URL of the JSP page, with the appended session ID.

# ACCESS THE SESSION ID

A session is started for each user who requests a JSP page from your Web site. Sessions enable a Web server to collect and use information entered by a user while the user accesses different resources on the Web server. For example, if a user specifies a user name on the main page of a Web site, this user name can be used by the Web server to personalize any other Web pages the user requests during that session. The Web server keeps track of each session by assigning a session ID that identifies each current user.

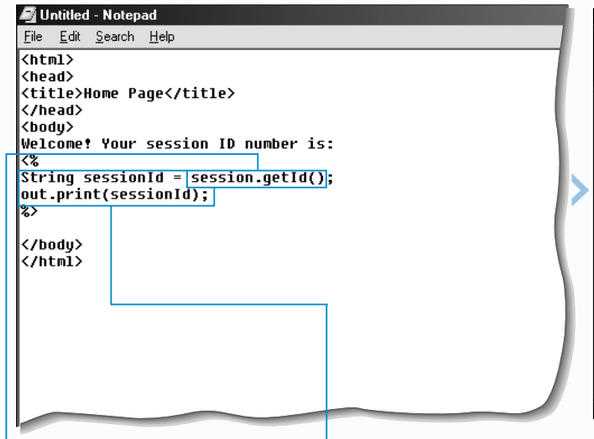
To access the session ID number, you can use the `getId` method of the `session` object. You cannot change a session ID you access. The format of the session ID will be different depending on the Web server you are using.

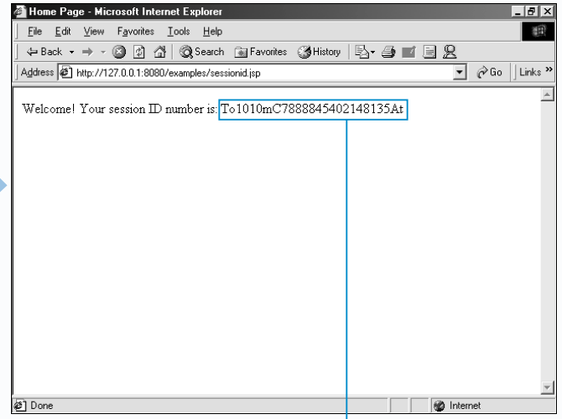
When a user requests a JSP page from your Web site, the Web server stores a session ID as a *cookie* on the user's computer. When the user requests another page from the site, the user's Web browser sends the session ID to the Web server to identify the user. If the user's Web browser or computer does not support cookies, you can use URL encoding to append the session ID to the URLs accessed by the user. For information about encoding URLs, see page 94.

A session ends when the user does not request another JSP page for a specific amount of time or when the session is *abandoned*. Any information that the Web server collected from the user during a session will be discarded when the session ends.

You should not use the session ID as the *primary key* in a database, as the session ID may not always be unique. For example, if the Web server is restarted, the server may assign a user a session ID that was previously assigned to a different user.

## ACCESS THE SESSION ID




**1** Type **session.getId()** where you want to access a session ID.

**2** Type the code that will display the session ID in a Web browser.

**3** Save the page with the .jsp extension and then display the JSP page in a Web browser.

■ The Web browser displays the result of accessing the session ID.





# ABANDON A SESSION

During a session, information is saved on the Web server and the client computer. As a result, each session requires the use of Web server resources, such as computer memory. If information for a session is no longer required, the session can be abandoned to free up resources on the Web server. This can improve the efficiency of a busy Web server.

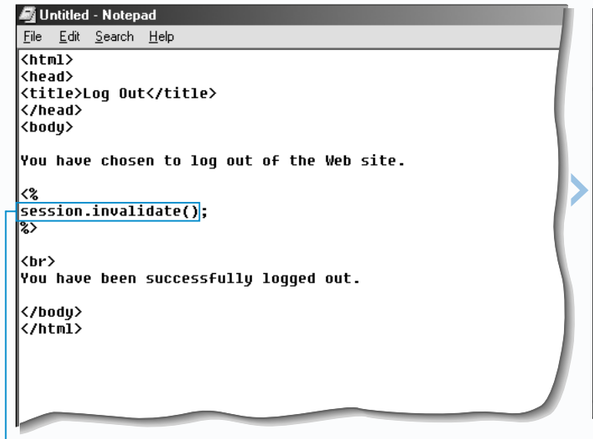
The `invalidate` method of the `session` object allows you to immediately end a session for one user and erase the information associated with the session. The information for the session will be permanently removed from the Web server. If you want to be able to later access the session information, you should write the information to a file or store the information in a database before abandoning the session.

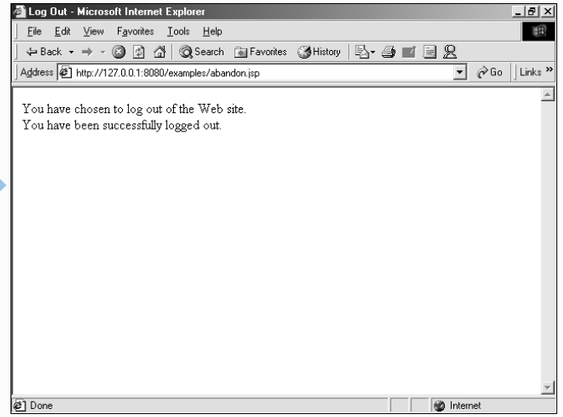
Abandoning a session is useful when an error occurs or when a user performs an action that indicates they no longer need the session information, such as logging out of the Web site. If the session was not abandoned, the Web server would keep the session information in memory until the session timed out. Abandoning a session also allows users to perform tasks such as clearing their Web site preferences or logging into your Web site using a different user name.

Abandoning a session does not stop the Web server from processing the JSP page, but does make session information generated before the session was abandoned unavailable to the page. An attempt to access session information after the session has been abandoned may generate an error.

Abandoning a session does not usually remove the cookie that stores session information on the client computer. The cookie will usually remain on the client until it is deleted by the Web browser, which typically occurs after the cookie expires or when a new session is started between the client and the Web server.

## ABANDON A SESSION




**1** Type **session.invalidate()** where you want to abandon a session.

**2** Save the page with the .jsp extension and then display the JSP page in a Web browser.

■ The Web server abandons the session.

# CREATE SESSION VALUES

As a user moves through the pages in your Web site, the user may be asked to enter information such as a user name, password or preferences to display each page. Creating session values allows you to store this information and make the information available to all the pages viewed by the user in your Web site. This saves the user from having to repeatedly enter the same information to display each page during a session.

You use the `setAttribute` method of the `session` object to create a session value. When creating a session value, you need to specify the name of the value and the information to be stored. A `null` value will be assigned if you do not specify any information for the session value. The information stored in a session value cannot be a primitive data type, such as `boolean` or `int`. For information on primitive data types, see page 30.

The information stored in a session value can come from sources such as forms, databases and cookies. The use of session values is an effective way of collecting and accessing information across multiple pages on a Web site and is more secure and easier to maintain than hidden fields or cookies.

All session values and the information stored in them will be discarded when the session ends or is terminated. If necessary, you can use cookies or a database to save the information stored in a session value.

After creating session values, the information stored in the session values can be accessed using the `getAttribute` method. For information about the `getAttribute` method, see page 100.

## Extra

You can turn off the use of session information for a JSP page by using the `page` directive. Turning off the use of session information does not produce any noticeable improvement in speed on the Web server, but it may offer increased security to JSP pages that do not use session information. If you try to use session values when session handling is turned off, an error will occur when the JSP page is viewed.

The `page` directive should be placed before any HTML code in a JSP page. To once again allow the use of session information in the JSP page, simply remove the `page` directive from the code.

**Example:**

```
<%@ page session = "false" %>
<html>
<head>
<title>Home Page</title>
</head>
<body>
<%
session.setAttribute("userName", "Tim");
session.setAttribute("preferredColor", "blue");
%>
</body>
</html>
```

## CREATE SESSION VALUES

Untitled - Notepad

```
<html>
<head>
<title>Session Values</title>
</head>
<body>

<%
session.setAttribute();

%>

Session values have been created.

</body>
</html>
```

**1** Type **session.setAttribute()** where you want to create a session value.

Untitled - Notepad

```
<html>
<head>
<title>Session Values</title>
</head>
<body>

<%
session.setAttribute("userName",);

%>

Session values have been created.

</body>
</html>
```

**2** Between the parentheses, type a name for the session value followed by a comma.

■ The name of the session value must be enclosed in quotation marks.

Untitled - Notepad

```
<html>
<head>
<title>Session Values</title>
</head>
<body>

<%
session.setAttribute("userName", "Sandy");
session.setAttribute("region", "Texas");
%>

Session values have been created.

</body>
</html>
```

**3** Type the information you want to store in the session value.

■ If you are storing a string, enclose the information in quotation marks.

**4** Repeat steps 1 to 3 for each session value you want to create.

Session Values - Microsoft Internet Explorer

Address http://127.0.0.1:8080/examples/sessionvalues.jsp

Session values have been created.

**5** Save the page with the .jsp extension and then display the JSP page in a Web browser.

■ You can now read the information stored in the session values. See page 100 to read session values.

# READ SESSION VALUES

If a JSP page in your Web site creates session values for a user, other JSP pages viewed by the user in the Web site can read and process the information stored in the session values until the session times out or is terminated. For information about creating session values, see page 98.

A Web server can personalize each JSP page in a Web site according to the user information saved in session values. For example, if a user prefers not to view images on Web pages, each page that the user visits in the Web site will read the session information for the user and display only text.

A JSP page reads the information stored in a session value using the `getAttribute` method of the `session` object. The JSP page that reads the information stored in a session value does not usually modify the information.

In most cases, the information stored in a session value is assigned to a variable. You can then use the variable to display the session information on the screen or to perform a more complex action, such as locating information in a database. You may have to `cast` the information stored in a session value as a data type that is compatible with the variable to which it is assigned. For information about casting, see page 30.

It is also important to note that variables can be accessed only by the JSP page on which they are created. If you want to use the same variable on another JSP page, you will have to recreate the variable and re-assign the information stored in the session value to the variable.

## Extra

In most cases, you know which session value you want to retrieve information from, but there may be times when you are required to find out which session values are available. You can use the `getAttributeNames` method of the `session` object to generate a list of the names of all the session values that are available during a session. You must cast the names as a collection that can be used by the `Iterator` interface. To use the `Iterator` interface, you must first import the interface from the `java.util` package.

TYPE THIS:

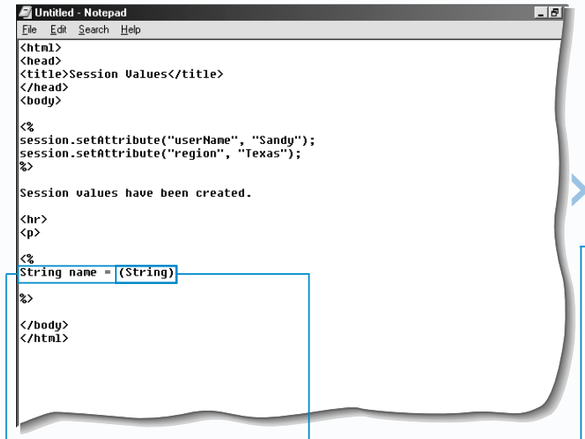
```
<body>
<%@ page import="java.util.Iterator" %>
Session values for this session:
<ul>
<%
Iterator sessionValues = (Iterator) session.getAttributeNames();

while (sessionValues.hasNext())
     out.print("<li>" + sessionValues.next() + "</li>");
%>
</ul>
</body>
```

RESULT:

Session values for this session:

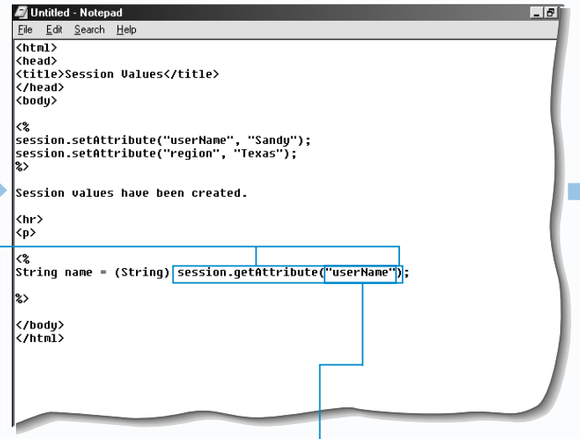
- login
- memberLevel
- region

## READ SESSION VALUES



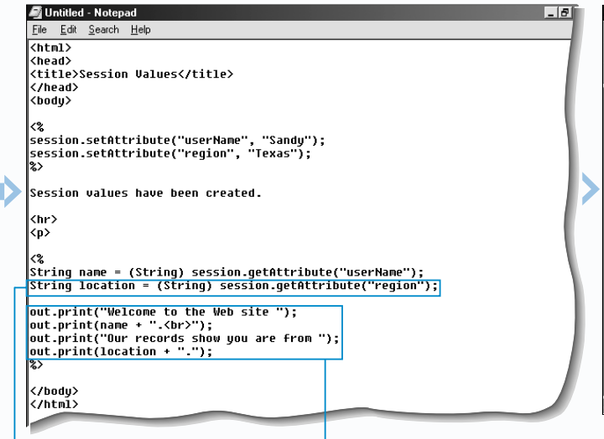

1 Type the code that declares a variable you want to store the information in a session value.

2 To cast the information in the session value as a specific data type, enter the data type you want to use, enclosed in parentheses.



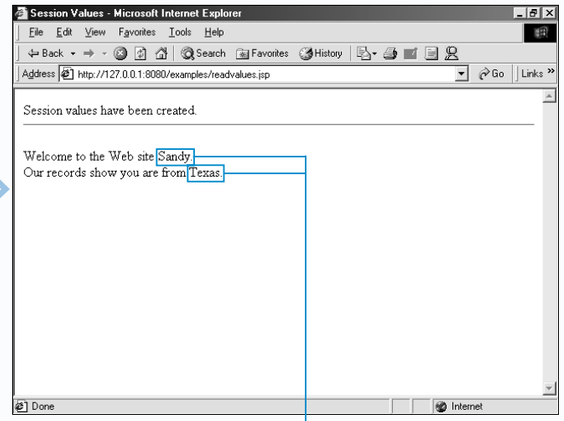

3 To read a session variable, type **session.getAttribute()**.

4 Between the parentheses, type the name of the session value you want to read, enclosed in quotation marks.




5 Repeat steps 1 to 4 for each session value you want to read and assign to a variable.

6 Type the code that uses the variables you created to store the information in the session values.




7 Save the page with the .jsp extension and then display the JSP page in a Web browser.

■ If the session values have been created, the Web browser displays the result of reading the session values.

# ADJUST THE SESSION TIMEOUT

The `setMaxInactiveInterval` method of the `session` object allows you to set the session timeout for a JSP page, in seconds. The session timeout determines how long a user's session information is stored on the Web server after the user last refreshes a page or requests a page in the Web site.

A session allows the Web server to identify a client computer as the user moves from page to page within a Web site. This is useful for applications such as shopping carts, when you need to be able to track the items a user has selected throughout your Web site. For more information about session information, see pages 96 to 101.

Typically, a user's session information is stored on the Web server for 30 minutes and is available to the JSP pages that the user views in your Web site. The session information created for a user will be available to the JSP pages in the Web site even if the user visits another Web site and then returns to your site within the timeout period. If the user returns to your Web site after the timeout period, the session information for the client will no longer be available.

The session timeout that you set for a JSP page applies to every client that accesses the JSP page.

Adjusting the session timeout period can help make your Web site more secure. For example, if you have a Web site that requires a user to log in, a short timeout period will help to prevent other users from accessing your site if the user leaves the computer while logged in. Keep in mind, however, that setting the session timeout too short may lead to the inadvertent loss of session information.

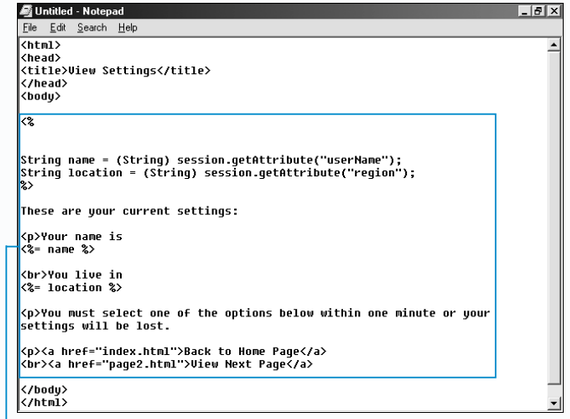
## Extra

The `session` object has many methods that can be used to alter or obtain information about the current session. Some methods and values of the `session` object can be accessed only during the session in which they were created. When the session ends, the items no longer exist. When another session starts, the items are recreated for that session.

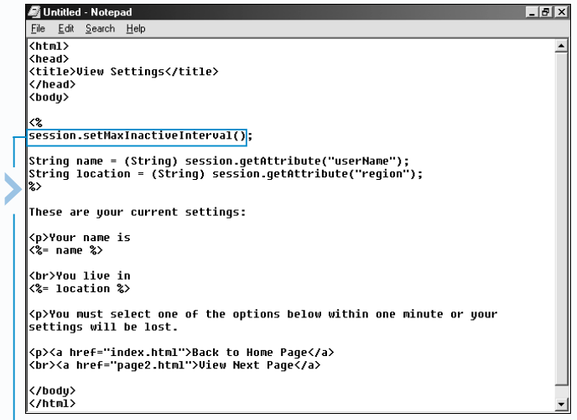
**Popular Session Object Methods**

| METHOD: | DESCRIPTION: |
|---|---|
| `getCreationTime` | Returns the time the session started, measured in milliseconds since January 1, 1970. |
| `getId` | Returns the session ID. |
| `getLastAccessedTime` | Returns the last time the client sent a request during the session, measured in milliseconds since January 1, 1970. |
| `setMaxInactiveInterval(interval)` | Sets the session timeout, in seconds. |
| `getMaxInactiveInterval` | Returns the session timeout, in seconds. |
| `invalidate` | Closes the session. |
| `isNew` | Returns `true` if the Web server has created a session, but the client computer has not yet accepted a session ID. |
| `getAttribute(name)` | Returns the information stored in a session value. |
| `getAttributeNames` | Returns a list of all session values. |
| `setAttribute(name, value)` | Creates a session value. |
| `removeAttribute(name)` | Removes a session value. |

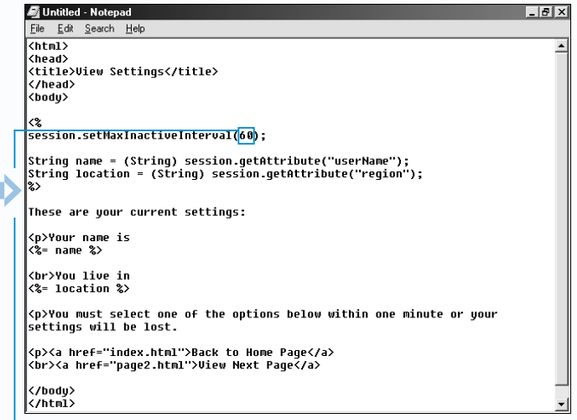
## ADJUST THE SESSION TIMEOUT



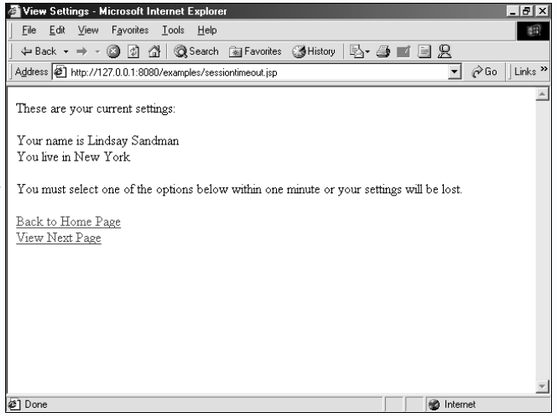

**1** Type the code you want to execute to display information in a Web browser.




**2** Type **session.setMaxInactiveInterval()** where you want to adjust the session timeout period for the JSP page.




**3** Between the parentheses, type the number of seconds you want the Web server to wait for activity before closing the session.




**4** Save the page with the .jsp extension and then display the JSP page in a Web browser.

- The Web browser displays the JSP page in which the session timeout is adjusted.
- If you do not request a new page in the Web site or refresh the page within the new timeout period, the Web server will erase your session information.

# USING APPLICATION VALUES

JavaServer Pages allows you to define a Web site or part of a Web site as an application. An application is a collection of JSP pages stored in a specific directory and its subdirectories on the Web server. For example, if you have 10 JSP pages stored in the same directory, those pages would make up an application.

All the JSP pages in an application typically must be stored in the same *virtual* directory on the Web server. The type of Web server you use will determine how the virtual directory and applications are created. For more information about creating virtual directories, refer to your Web server documentation.

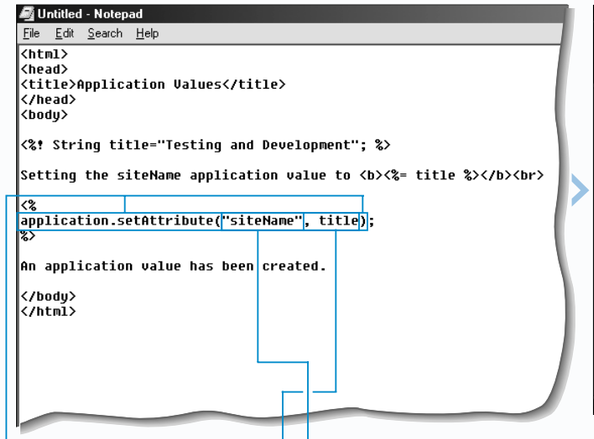
You use the `setAttribute` method of the `application` object to create an application value. When using the `setAttribute` method, you must specify the name of the application value and the information the value will contain. The information stored in an application value cannot be a primitive data type, such as `boolean` or `int`. For information about primitive data types, see page 30.

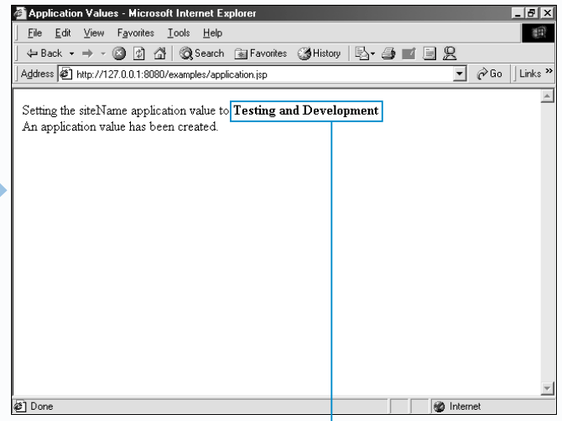
All the JSP pages in an application can access the information stored in an application value. For example, if you create an application value that stores a counter, the number of people who have used your application could be displayed at the bottom of each page in the application.

You access an application value in your JSP pages using the `getAttribute` method of the `application` object. If a JSP page tries to access an application value that does not exist, the `getAttribute` method will return a value of null.

An application starts when the first user requests a JSP page from the application and ends when the Web server shuts down or restarts. Application values are discarded when the application ends.

## CREATE AN APPLICATION VALUE




**1** Type **application.setAttribute()** where you want to create an application value.

**2** Between the parentheses, type a name for the application value, enclosed in quotation marks.

**3** Type a comma followed by the information you want the application value to use.

**4** Save the page with the .jsp extension and then display the JSP page in a Web browser.

■ The Web server activates the application value. You can now access the information stored in the application value.





## Extra

You can delete an existing application value using the `removeAttribute` method. You should delete any application values that you no longer need. If a JSP page tries to access an application value that has been removed, a null value will be generated.

**TYPE THIS:**

```
Welcome to the <b>
<%= application.getAttribute("siteName") %>
</b> Web site. <br>
<% application.removeAttribute("siteName"); %>
Application value deleted. <br>
Welcome to the <b>
<%= application.getAttribute("siteName") %>
</b> Web site.
```

**RESULT:**

Welcome to the **Testing and Development** Web site.
Application value deleted.
Welcome to the **null** Web site.

You can change the information stored in an application value. If the application value has not yet been created, changing the information stored in the value will create the value.

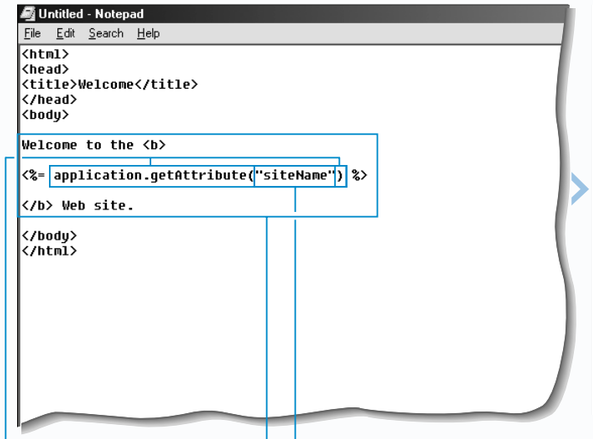
**TYPE THIS:**

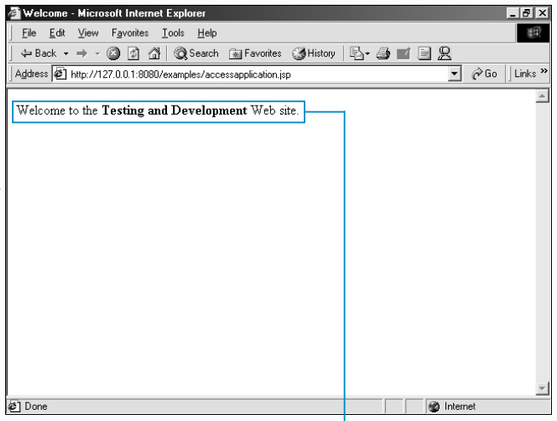
```
Old Web site name: <b>
<%= application.getAttribute("siteName") %></b><br>
<% application.setAttribute("siteName",
"ABC Corporation"); %>
New Web site name: <b>
<%= application.getAttribute("siteName") %></b>
```

**RESULT:**

Old Web site name: **Testing and Development**
New Web site name: **ABC Corporation**

## ACCESS AN APPLICATION VALUE




**1** Type **application.getAttribute()** where you want to access an application value.

**2** Between the parentheses, type the name of the application value you want to access, enclosed in quotation marks.

**3** Type the code that uses the application value.

**4** Save the page with the .jsp extension and then display the JSP page in a Web browser.

*Note: You must save the page in the same directory that stores the JSP page in which the application value was created.*

■ The Web browser displays the result of accessing the application value.

# DETERMINE THE PATH OF A FILE

The `getRealPath` method of the `application` object allows you to identify where a file, such as a Web page or JSP page, is stored on the Web server.

A Web server can store files in many different directories. The directory that stores a page is not always apparent in the URL of the page. For example, a JSP page named login.jsp stored in the directory C:\Tomcat\webapps\public\sign_in could have the URL http://www.abccorp.com/sign_in/login.jsp. When a JSP page needs to access a page on the Web server, such as when using the `include` directive to access information from a Web page, the JSP page may need to know the exact location of the page, not the URL of the page.

To identify the path of a page, you must know the filename of the page. The `getRealPath` method uses the filename of the page, enclosed in quotation marks, as its argument.

Regardless of the operating system you use, you should use slashes (/) within the path of the page you want to locate. When the path to the page starts with a slash (/), the path will be determined starting at the document root directory of the current Web application. The document root directory is the parent directory that contains all the documents and applications on a Web server. The location of the document root directory depends on the configuration of the Web server. On Web servers that host multiple Web sites, the document root directory will be different for each Web site.

The result returned by the `getRealPath` method is a string value. You can assign this value to a variable and then use the variable in your code.

The `getRealPath` method shows where a page is located on the Web server but does not verify that the page or the directories actually exist.

## Apply It

You can determine the path of the current JSP page by using a single slash enclosed in quotation marks as the argument for the `getRealPath` method. Identifying the path of the current page is useful when you are creating a JSP page for different Web applications and you need to make sure the directory structure is the same for each application.

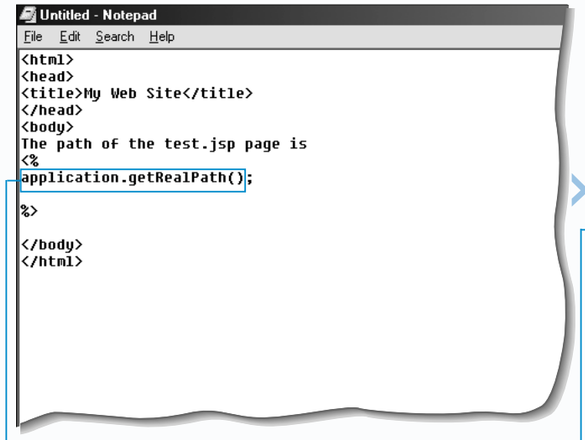
TYPE THIS:

```
<html>
<head>
<title>My Web Site</title>
</head>
<body>
This JSP page is stored in
<%
String docPath = application.getRealPath("/");
out.print(docPath);
%>

</body>
</html>
```

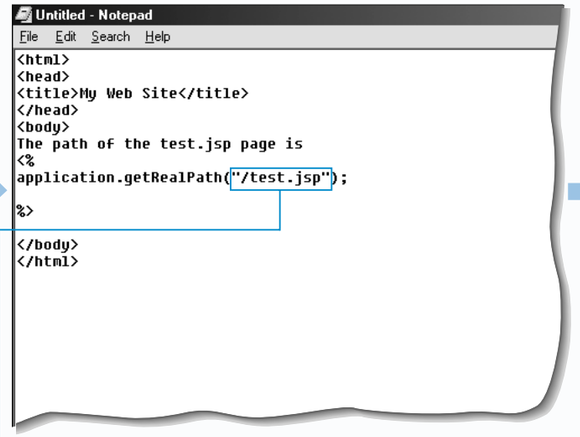
RESULT:

This JSP page is stored in C:\Tomcat\webapps\examples\

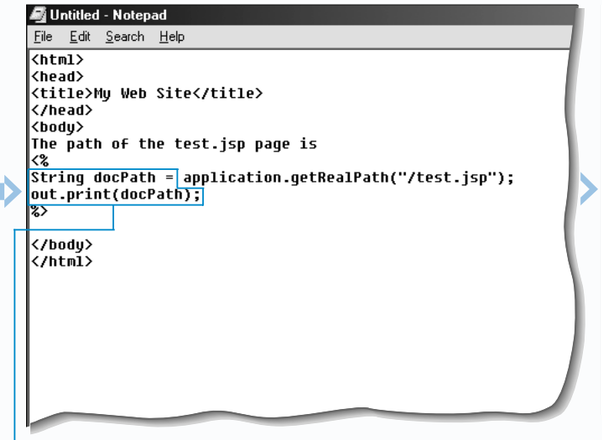
## DETERMINE THE PATH OF A FILE



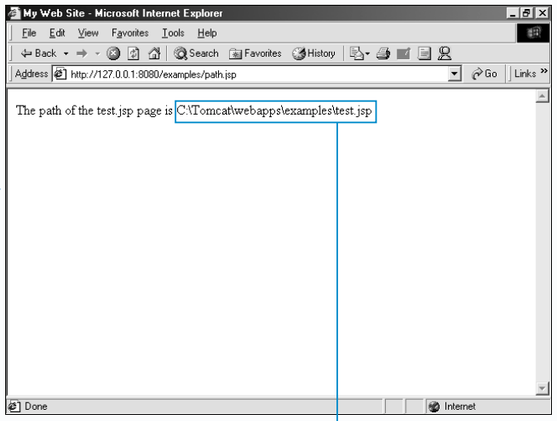

1 Type **application.getRealPath()** where you want to find the path of a file.




2 Between the parentheses, type a slash (/) followed by the name of the file whose path you want to determine, enclosed in quotation marks.




3 Type the code that will display the path information in a Web browser.




4 Save the page with the .jsp extension and then display the JSP page in a Web browser.

■ The Web browser displays the result of determining the path of a file.

# GENERATE A NEWLINE CHARACTER

A newline character instructs a processing program to stop placing output on the current line and begin a new line. The `newLine` method of the `out` object can be used in a JSP page to generate a newline character. Newline characters are sometimes called line separators.

To generate a newline character, you create a scriptlet that contains the `out.newLine` statement. Scriptlets are processed by the Web server and a newline character generated by a scriptlet is inserted into the source code for a JSP page before the page is displayed.

Since Web browsers ignore extra spaces and new lines in source code, the line break you add using a newline character will not appear on a JSP page when the page is displayed in a Web browser. To view the results of generating a newline character, you must display the source code for your JSP page. Most Web browsers allow users to easily view the source code for a page. A new line will begin in the source code where you added the newline character. To have a new line appear on your JSP page when it is displayed in a Web browser, use the HTML tag `<br>`.

Using newline characters is particularly useful when a JSP page generates HTML source code. HTML code that does not contain any new lines can be difficult to read and troubleshoot. By inserting new lines into the code, you can separate the various elements on the page, making the page easier to understand. For example, a page containing images and text can have newline characters after each paragraph and image. Newline characters are typically inserted after closing HTML tags, such as the `</p>` and `</img>` tags.

## Apply It

The actual character or characters a computer uses for a new line depends on the operating system installed on the computer. For example, a new line may be created by a carriage return, a newline character or both. Because these characters are not displayable, you cannot view them. You can, however, use the Java `getBytes` method to view the ASCII code for the characters.

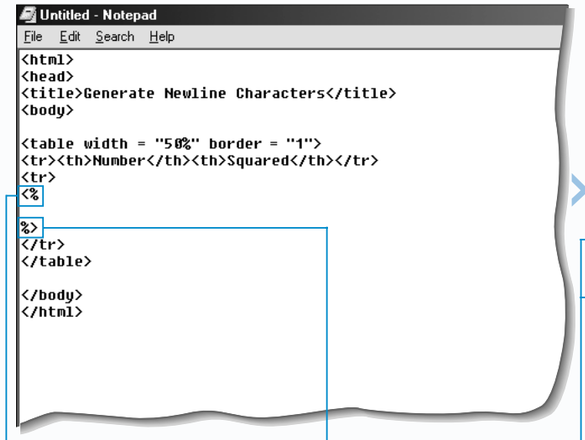
**TYPE THIS:**

```
The ASCII codes for the characters used to create new lines on this computer are: <br>
<%
String lineSeparator = System.getProperties().getProperty("line.separator");
byte[] array = lineSeparator.getBytes();

for (int x = 0; x < array.length; x++)
     out.print(array[x] + "<br>");
%>
```

**RESULT:**

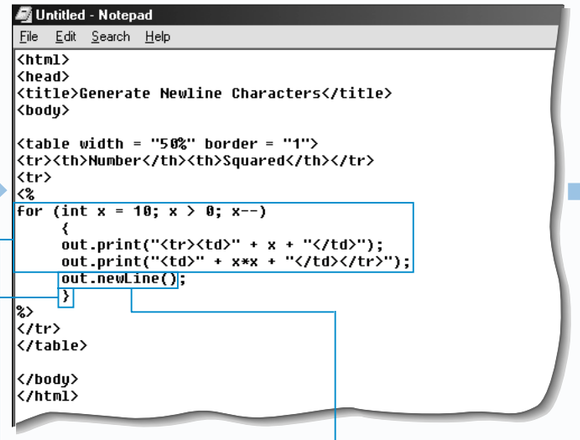
The ASCII codes for the characters used to create new lines on this computer are:
13
10

## GENERATE A NEWLINE CHARACTER



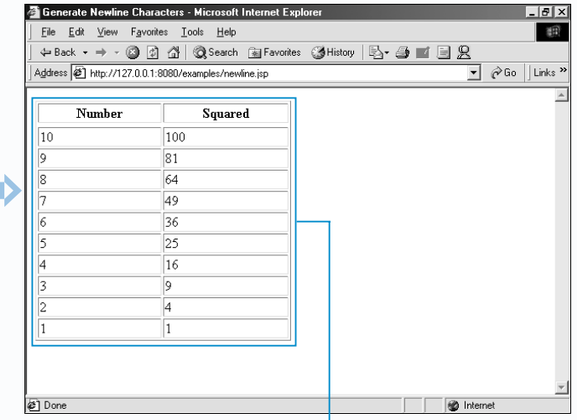

**1** Type **<%** where you want the scriptlet that will generate a newline character to begin.

**2** Type **%>** where you want the scriptlet to end.



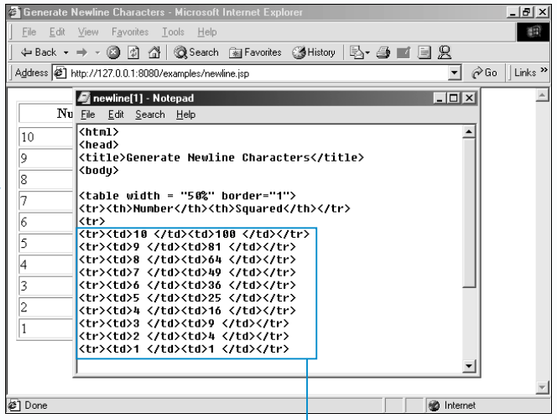

**3** Type the code that will generate HTML source code for the JSP page.

**4** Type **out.newLine( )** where you want to generate a newline character.




**5** Save the page with the .jsp extension and then display the JSP page in a Web browser.

■ Generating newline characters does not affect the way a JSP page appears in a Web browser.




**6** Display the source code for the JSP page.

■ The source code displays the result of generating newline characters.

# DETERMINE THE OPERATING SYSTEM

You can use the `getProperties` and `getProperty` methods of the `System` object to determine the operating system being used on the computer running your JSP pages.

JSP code should not have problems running on different operating systems, but the way JSP interacts with the computer may differ depending on the operating system. For example, you may develop JSP pages on a computer using a Windows operating system and then transfer the JSP pages to an Internet Web server that uses the Linux operating system. When JSP pages run on a computer using a Windows operating system, the JSP pages will attempt to find files, such as include files, in a specific directory. When the JSP pages run on a computer using the Linux operating system, errors may occur because the required files may be located in a different directory. Instead of creating two sets of JSP pages, you could simply set the JSP page to determine which operating system is running on the computer and then automatically alter the path required to access the files.

The `getProperties` method returns all of the system properties that are specific to the computer running the JSP pages. The system properties that are available depend on the operating system running on the computer.

You use the `getProperty` method to specify the name of a specific property you wish to access. The property name used to identify the current operating system is `os.name`.

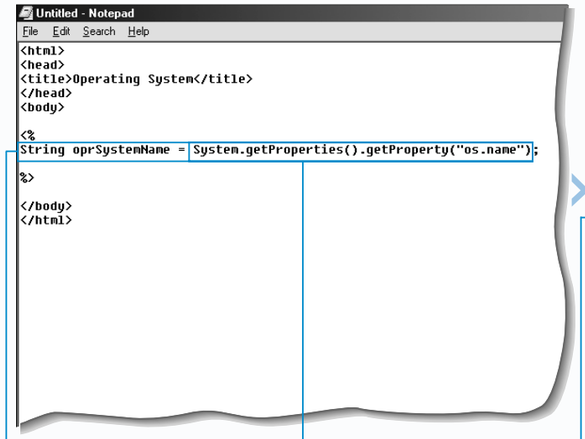
The value returned by the `getProperties` and `getProperty` methods is a `String` data type and can be assigned to a variable, which can then be used in your code. You can use the `indexOf` method to match the content of the variable with the name of a specific operating system. Refer to the Java SDK documentation for more information about using the `indexOf` method.

## Extra

The following is a list of some of the other system properties you may determine using the `getProperty` method:

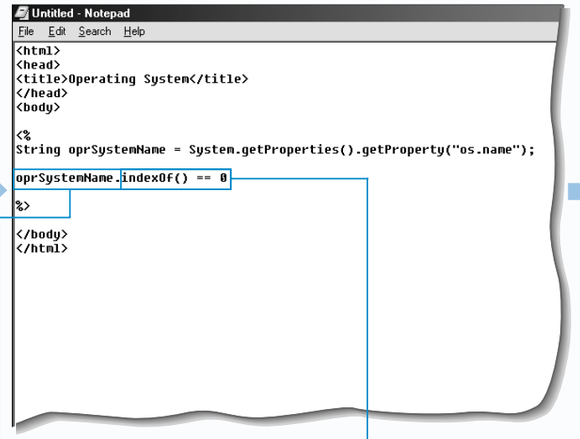
| PROPERTY NAME: | RETURNS: |
|---|---|
| java.home | The directory where Java is installed |
| java.class.path | The path where the classes are loaded from |
| java.version | The version of the Java API implementation |
| java.vendor | The vendor of the Java API implementation |
| java.class.version | The version of the Java class file format |
| os.arch | The architecture of the operating system |
| os.version | The version of the operating system |
| user.name | The account name of the current user |
| user.home | The home directory of the current user |
| user.dir | The current working directory |

## DETERMINE THE OPERATING SYSTEM



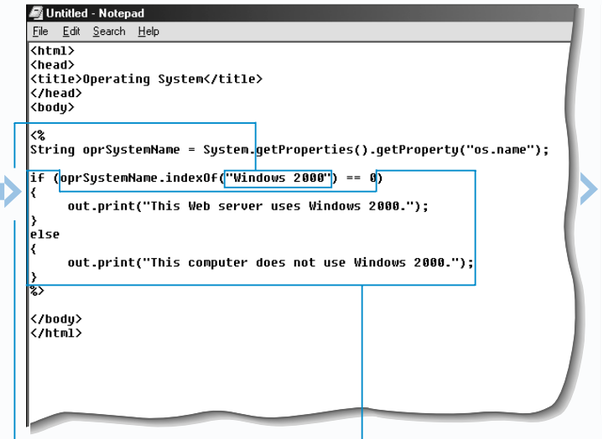

1 Type the code that declares a variable you want to store the name of the operating system.

2 Type **System.getProperties().getProperty("os.name")** to determine the name of the operating system.



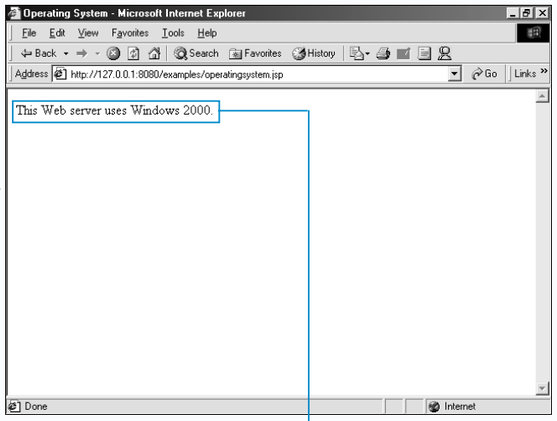

3 To match the content of the variable that stores the name of the operating system with the name of a specific operating system, type the name of the variable followed by a dot.

4 Type **indexOf() == 0**.




5 Between the parentheses, type the name of the operating system you want to check for, enclosed in quotation marks.

6 Type the code that uses the information retrieved by the `getProperties` and `getProperty` methods.




7 Save the page with the .jsp extension and then display the JSP page in a Web browser.

■ The Web browser displays the result of determining the operating system running on the computer.

# FORWARD TO ANOTHER JSP PAGE

The <jsp:forward> tag is used to instruct a Web server to stop processing the current JSP page and start processing another page. For example, when an error occurs during the processing of a JSP page, you can use the <jsp:forward> tag to transfer control to another JSP page that handles errors and displays help information for the user. The <jsp:forward> tag is also useful for transferring control to a different JSP page depending on the value of a variable, such as a user name or the time of day.

When using the <jsp:forward> tag, you assign a value to the page attribute. The value can be a string literal, a value generated by an expression or the relative path of the JSP page that control will be transferred to.

When the Web server processes a JSP page that contains a <jsp:forward> tag, the server stops processing the page and executes the code in the JSP page specified in the tag. The Web server does not return to the original page.

You should use the <jsp:forward> tag early in your code. No information should be sent to the client before the <jsp:forward> tag is executed or an error will be generated. Any data currently in the buffer when the <jsp:forward> tag is encountered will be deleted.

Any information available to the original JSP page will also be available to the JSP page that control is transferred to. Information available to the controlling JSP page includes application values, session values and any data stored in a request object, such as values submitted to a form. The JSP page control is transferred to can access this information even if the page is not part of the same application as the original JSP page.

## Extra

The <jsp:param> tag can be used to pass additional information to the request object before transferring control to the other JSP page. For example, you can use the <jsp:param> tag to create a parameter that stores the name of the page that forwarded the request object. This allows the page that will receive control to use the getParameter method of the request object to determine where the request object originated. The <jsp:param> tag is placed between the <jsp:forward> and the </jsp:forward> tags.

**Type this in the original JSP page:**

```
<jsp:forward page="logout.jsp">
<jsp:param name="callingPage" value="index.jsp"/>
</jsp:forward>
```

**Type this in the page control is being transferred to:**

```
You are now logged out.<br>
You have been forwarded to this page from the JSP page:<br>
<%= request.getParameter("callingPage") %>
```

**Result:**

You are now logged out.
You have been forwarded to this page from the JSP page:
index.jsp

## FORWARD TO ANOTHER JSP PAGE

Untitled - Notepad

```
<html>
<head>
<title>Logged Out</title>
</head>
<body>

You have been logged out.

</body>
</html>
```

**CREATE A JSP PAGE YOU WANT TO FORWARD TO**

1 In a text editor, create the JSP page you want to transfer control to.

2 Save the page on the Web server with the .jsp extension.

Untitled - Notepad

```
<html>
<head>
<title>Welcome</title>
</head>
<body>

Welcome to my Web page.

<%
String loggedOutPage = "page2.jsp";
%>

</body>
</html>
```

**FORWARD TO ANOTHER JSP PAGE**

1 Display the page in which you want to transfer control to another JSP page.

2 To create a variable that will store the parameter for the JSP page you want to transfer control to, type **String**, followed by a name for the variable.

3 Type **=** followed by a value for the variable, enclosed in quotation marks.

Untitled - Notepad

```
<html>
<head>
<title>Welcome</title>
</head>
<body>

Welcome to my Web page.

<%
String loggedOutPage = "page2.jsp";
%>

<jsp:forward page="<%= loggedOutPage %>" />

</body>
</html>
```

4 Type **<jsp:forward page=** where you want to transfer control to another page.

5 Type the expression that generates the name of the page you want to transfer control to, enclosed in quotation marks.

*Note: You can also type the path of the page or a string literal.*

6 Type **/>** to complete the tag.

Logged Out - Microsoft Internet Explorer

Address http://127.0.0.1:8080/examples/forward.jsp

You have been logged out.

7 Save the page with the .jsp extension and then display the JSP page in a Web browser.

■ The Web browser displays the results of forwarding control to another JSP page.