**JSP** technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.

A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.

**Advantages of JSP over Servlet**

There are many advantages of JSP over the Servlet. They are as follows:

**1) Extension to Servlet**

JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

#### 2) Easy to maintain

JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.

#### 3) Fast Development: No need to recompile and redeploy

If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

#### 4) Less code than Servlet

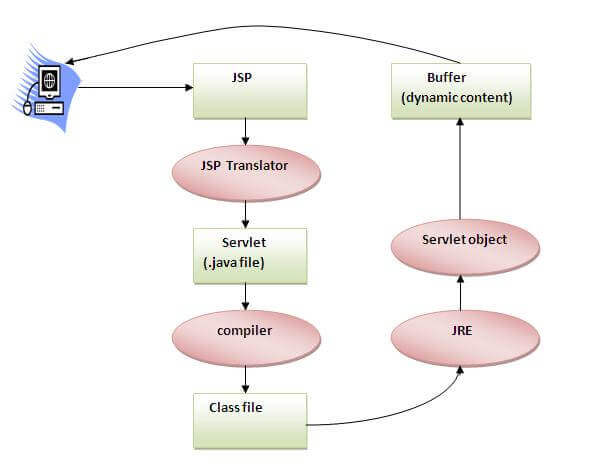
In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

**The Lifecycle of a JSP Page**

The JSP pages follow these phases:

* Translation of JSP Page
* Compilation of JSP Page
* Classloading (the classloader loads class file)
* Instantiation (Object of the Generated Servlet is created).
* Initialization ( the container invokes jspInit() method).
* Request processing ( the container invokes \_jspService() method).
* Destroy ( the container invokes jspDestroy() method).

**Note: jspInit(), \_jspService() and jspDestroy() are the life cycle methods of JSP.**



As depicted in the above diagram, JSP page is translated into Servlet by the help of JSP translator. The JSP translator is a part of the web server which is responsible for translating the JSP page into Servlet. After that, Servlet page is compiled by the compiler and gets converted into the class file. Moreover, all the processes that happen in Servlet are performed on JSP later like initialization, committing response to the browser and destroy.

**Creating a simple JSP Page**

To create the first JSP page, write some HTML code as given below, and save it by .jsp extension. We have saved this file as index.jsp. Put it in a folder and paste the folder in the web-apps directory in apache tomcat to run the JSP page.

**index.jsp**

Let's see the simple example of JSP where we are using the scriptlet tag to put Java code in the JSP page. We will learn scriptlet tag later.

1. <html>
2. <body>
3. <% out.print(2\*5); %>
4. </body>
5. </html>

It will print **10** on the browser.

**How to run a simple JSP Page?**

Follow the following steps to execute this JSP page:

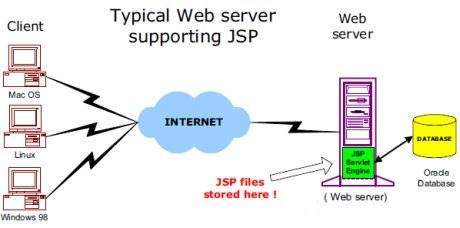
* Start the server
* Put the JSP file in a folder and deploy on the server
* Visit the browser by the URL http://localhost:portno/contextRoot/jspfile, for example, http://localhost:8888/myapplication/index.jsp

# JSP - Architecture

The web server needs a JSP engine, i.e, a container to process JSP pages. The JSP container is responsible for intercepting requests for JSP pages. This tutorial makes use of Apache which has built-in JSP container to support JSP pages development.

A JSP container works with the Web server to provide the runtime environment and other services a JSP needs. It knows how to understand the special elements that are part of JSPs.

Following diagram shows the position of JSP container and JSP files in a Web application.

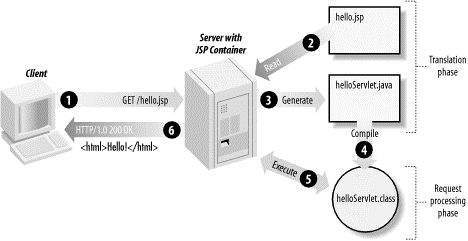


## JSP Processing

The following steps explain how the web server creates the Webpage using JSP −

* As with a normal page, your browser sends an HTTP request to the web server.
* The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with **.jsp** instead of **.html**.
* The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to println( ) statements and all JSP elements are converted to Java code. This code implements the corresponding dynamic behavior of the page.
* The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
* A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format. The output is furthur passed on to the web server by the servlet engine inside an HTTP response.
* The web server forwards the HTTP response to your browser in terms of static HTML content.
* Finally, the web browser handles the dynamically-generated HTML page inside the HTTP response exactly as if it were a static page.

All the above mentioned steps can be seen in the following diagram −



Typically, the JSP engine checks to see whether a servlet for a JSP file already exists and whether the modification date on the JSP is older than the servlet. If the JSP is older than its generated servlet, the JSP container assumes that the JSP hasn't changed and that the generated servlet still matches the JSP's contents. This makes the process more efficient than with the other scripting languages (such as PHP) and therefore faster.

So in a way, a JSP page is really just another way to write a servlet without having to be a Java programming wiz. Except for the translation phase, a JSP page is handled exactly like a regular servlet.

**JSP Directives**

The **jsp directives** are messages that tells the web container how to translate a JSP page into the corresponding servlet.

There are three types of directives:

* page directive
* include directive
* taglib directive

**Syntax of JSP Directive**

1. <%@ directive attribute="value" %>

**JSP page directive**

The page directive defines attributes that apply to an entire JSP page.

**Syntax of JSP page directive**

1. <%@ page attribute="value" %>

**Attributes of JSP page directive**

* import
* contentType
* extends
* info
* buffer
* language
* isELIgnored
* isThreadSafe
* autoFlush
* session
* pageEncoding
* errorPage
* isErrorPage
* **1)import**

|  |
| --- |
| The import attribute is used to import class,interface or all the members of a package.It is similar to import keyword in java class or interface. |

**Example of import attribute**

1. <html>
2. <body>
4. <%@ page import="java.util.Date" %>
5. Today is: <%= new Date() %>
7. </body>
8. </html>

**2)contentType**

The contentType attribute defines the MIME(Multipurpose Internet Mail Extension) type of the HTTP response.The default value is "text/html;charset=ISO-8859-1".

**Example of contentType attribute**

1. <html>
2. <body>
4. <%@ page contentType=application/msword %>
5. Today is: <%= new java.util.Date() %>
7. </body>
8. </html>

**3)extends**

The extends attribute defines the parent class that will be inherited by the generated servlet.It is rarely used.

**4)info**

This attribute simply sets the information of the JSP page which is retrieved later by using getServletInfo() method of Servlet interface.

**Example of info attribute**

1. <html>
2. <body>
4. <%@ page info="composed by Sonoo Jaiswal" %>
5. Today is: <%= new java.util.Date() %>
7. </body>
8. </html>

The web container will create a method getServletInfo() in the resulting servlet.For example:

1. public String getServletInfo() {
2. return "composed by Sonoo Jaiswal";
3. }

**5)buffer**

The buffer attribute sets the buffer size in kilobytes to handle output generated by the JSP page.The default size of the buffer is 8Kb.

**Example of buffer attribute**

1. <html>
2. <body>
4. <%@ page buffer="16kb" %>
5. Today is: <%= new java.util.Date() %>
7. </body>
8. </html>

**6)language**

The language attribute specifies the scripting language used in the JSP page. The default value is "java".

**7)isELIgnored**

|  |
| --- |
| We can ignore the Expression Language (EL) in jsp by the isELIgnored attribute. By default its value is false i.e. Expression Language is enabled by default. We see Expression Language later. |

1. <%@ page isELIgnored="true" %>//Now EL will be ignored

**8)isThreadSafe**

|  |
| --- |
| Servlet and JSP both are multithreaded.If you want to control this behaviour of JSP page, you can use isThreadSafe attribute of page directive.The value of isThreadSafe value is true.If you make it false, the web container will serialize the multiple requests, i.e. it will wait until the JSP finishes responding to a request before passing another request to it.If you make the value of isThreadSafe attribute like: |

<%@ page isThreadSafe="false" %>

The web container in such a case, will generate the servlet as:

1. public class SimplePage\_jsp extends HttpJspBase
2. implements SingleThreadModel{
3. .......
4. }

**9)errorPage**

The errorPage attribute is used to define the error page, if exception occurs in the current page, it will be redirected to the error page.

**Example of errorPage attribute**

1. //index.jsp
2. <html>
3. <body>
5. <%@ page errorPage="myerrorpage.jsp" %>
7. <%= 100/0 %>
9. </body>
10. </html>

**10)isErrorPage**

The isErrorPage attribute is used to declare that the current page is the error page.

**Note: The exception object can only be used in the error page.**

**Example of isErrorPage attribute**

1. //myerrorpage.jsp
2. <html>
3. <body>
5. <%@ page isErrorPage="true" %>
7. Sorry an exception occured!<br/>
8. The exception is: <%= exception %>
10. </body>
11. </html>

# Jsp Include Directive

The include directive is used to include the contents of any resource it may be jsp file, html file or text file. The include directive includes the original content of the included resource at page translation time (the jsp page is translated only once so it will be better to include static resource).

### Advantage of Include directive

Code Reusability

### Syntax of include directive

1. <%@ include file="resourceName" %>

### Example of include directive

In this example, we are including the content of the header.html file. To run this example you must create an header.html file.

1. <html>
2. <body>
4. <%@ include file="header.html" %>
6. Today is: <%= java.util.Calendar.getInstance().getTime() %>
8. </body>
9. </html>

#### Note: The include directive includes the original content, so the actual page size grows at runtime.

# JSP Taglib directive

The JSP taglib directive is used to define a tag library that defines many tags. We use the TLD (Tag Library Descriptor) file to define the tags.

#### Syntax JSP Taglib directive

1. <%@ taglib uri="uriofthetaglibrary" prefix="prefixoftaglibrary" %>

### Example of JSP Taglib directive

In this example, we are using our tag named currentDate. To use this tag we must specify the taglib directive so the container may get information about the tag.

1. <html>
2. <body>
4. <%@ taglib uri="http://www.javatpoint.com/tags" prefix="mytag" %>
6. <mytag:currentDate/>
8. </body>
9. </html>

# JSP Scriptlet tag (Scripting elements)

In JSP, java code can be written inside the jsp page using the scriptlet tag. Let's see what are the scripting elements first.

## JSP Scripting elements

The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:

* scriptlet tag
* expression tag
* declaration tag

### JSP scriptlet tag

A scriptlet tag is used to execute java source code in JSP. Syntax is as follows:

1. <%  java source code %>

### Example of JSP scriptlet tag

In this example, we are displaying a welcome message.

1. <html>
2. <body>
3. <% out.print("welcome to jsp"); %>
4. </body>
5. </html>

### Example of JSP scriptlet tag that prints the user name

In this example, we have created two files index.html and welcome.jsp. The index.html file gets the username from the user and the welcome.jsp file prints the username with the welcome message.

File: index.html

1. <html>
2. <body>
3. <form action="welcome.jsp">
4. <input type="text" name="uname">
5. <input type="submit" value="go"><br/>
6. </form>
7. </body>
8. </html>

File: welcome.jsp

# SP expression tag

The code placed within **JSP expression tag** is written to the output stream of the response. So you need not write out.print() to write data. It is mainly used to print the values of variable or method.

### Syntax of JSP expression tag

1. <%=  statement %>

### Example of JSP expression tag

In this example of jsp expression tag, we are simply displaying a welcome message.

1. <html>
2. <body>
3. <%= "welcome to jsp" %>
4. </body>
5. </html>

#### Note: Do not end your statement with semicolon in case of expression tag.

### Example of JSP expression tag that prints current time

To display the current time, we have used the getTime() method of Calendar class. The getTime() is an instance method of Calendar class, so we have called it after getting the instance of Calendar class by the getInstance() method.

index.jsp

1. <html>
2. <body>
3. Current Time: <%= java.util.Calendar.getInstance().getTime() %>
4. </body>
5. </html>

### Example of JSP expression tag that prints the user name

In this example, we are printing the username using the expression tag. The index.html file gets the username and sends the request to the welcome.jsp file, which displays the username.

File: index.jsp

1. <html>
2. <body>
3. <form action="welcome.jsp">
4. <input type="text" name="uname"><br/>
5. <input type="submit" value="go">
6. </form>
7. </body>
8. </html>

File: welcome.jsp

1. <html>
2. <body>
3. <%= "Welcome "+request.getParameter("uname") %>
4. </body>
5. </html>

# JSP Declaration Tag

The **JSP declaration tag** is used to declare fields and methods.

The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet.

So it doesn't get memory at each request.

#### Syntax of JSP declaration tag

The syntax of the declaration tag is as follows:

1. <%!  field or method declaration %>

### Difference between JSP Scriptlet tag and Declaration tag

|  |  |
| --- | --- |
| **Jsp Scriptlet Tag** | **Jsp Declaration Tag** |
| The jsp scriptlet tag can only declare variables not methods. | The jsp declaration tag can declare variables as well as methods. |
| The declaration of scriptlet tag is placed inside the \_jspService() method. | The declaration of jsp declaration tag is placed outside the \_jspService() method. |

### Example of JSP declaration tag that declares field

In this example of JSP declaration tag, we are declaring the field and printing the value of the declared field using the jsp expression tag.

### index.jsp

1. <html>
2. <body>
3. <%! int data=50; %>
4. <%= "Value of the variable is:"+data %>
5. </body>
6. </html>

### Example of JSP declaration tag that declares method

In this example of JSP declaration tag, we are defining the method which returns the cube of given number and calling this method from the jsp expression tag. But we can also use jsp scriptlet tag to call the declared method.

### index.jsp

1. <html>
2. <body>
3. <%!
4. int cube(int n){
5. return n\*n\*n\*;
6. }
7. %>
8. <%= "Cube of 3 is:"+cube(3) %>
9. </body>
10. </html>

# JSP Action Tags

There are many JSP action tags or elements. Each JSP action tag is used to perform some specific tasks.

The action tags are used to control the flow between pages and to use Java Bean. The Jsp action tags are given below.

|  |  |
| --- | --- |
| **JSP Action Tags** | **Description** |
| jsp:forward | forwards the request and response to another resource. |
| jsp:include | includes another resource. |
| jsp:useBean | creates or locates bean object. |
| jsp:setProperty | sets the value of property in bean object. |
| jsp:getProperty | prints the value of property of the bean. |
| jsp:plugin | embeds another components such as applet. |
| jsp:param | sets the parameter value. It is used in forward and include mostly. |
| jsp:fallback | can be used to print the message if plugin is working. It is used in jsp:plugin. |

The jsp:useBean, jsp:setProperty and jsp:getProperty tags are used for bean development. So we will see these tags in bean developement.

**jsp:forward action tag**

The jsp:forward action tag is used to forward the request to another resource it may be jsp, html or another resource.

**Syntax of jsp:forward action tag without parameter**

1. <jsp:forward page="relativeURL | <%= expression %>" />

**Syntax of jsp:forward action tag with parameter**

1. <jsp:forward page="relativeURL | <%= expression %>">
2. <jsp:param name="parametername" value="parametervalue | <%=expression%>" />
3. </jsp:forward>

**Example of jsp:forward action tag without parameter**

In this example, we are simply forwarding the request to the printdate.jsp file.

**index.jsp**

1. <html>
2. <body>
3. <h2>this is index page</h2>
5. <jsp:forward page="printdate.jsp" />
6. </body>
7. </html>

**printdate.jsp**

1. <html>
2. <body>
3. <% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
4. </body>
5. </html>

**Difference between jsp include directive and include action**

|  |  |
| --- | --- |
| **JSP include directive** | **JSP include action** |
| includes resource at translation time. | includes resource at request time. |
| better for static pages. | better for dynamic pages. |
| includes the original content in the generated servlet. | calls the include method. |

**Syntax of jsp:include action tag without parameter**

1. <jsp:include page="relativeURL | <%= expression %>" />

**Syntax of jsp:include action tag with parameter**

1. <jsp:include page="relativeURL | <%= expression %>">
2. <jsp:param name="parametername" value="parametervalue | <%=expression%>" />
3. </jsp:include>

# JavaBean

A JavaBean is a Java class that should follow the following conventions:

* It should have a no-arg constructor.
* It should be Serializable.
* It should provide methods to set and get the values of the properties, known as getter and setter methods.

## Why use JavaBean?

According to Java white paper, it is a reusable software component. A bean encapsulates many objects into one object so that we can access this object from multiple places. Moreover, it provides easy maintenance.

## Simple example of JavaBean class

1. //Employee.java
3. package mypack;
4. public class Employee implements java.io.Serializable{
5. private int id;
6. private String name;
7. public Employee(){}
8. public void setId(int id){this.id=id;}
9. public int getId(){return id;}
10. public void setName(String name){this.name=name;}
11. public String getName(){return name;}
12. }

## How to access the JavaBean class?

To access the JavaBean class, we should use getter and setter methods.

1. package mypack;
2. public class Test{
3. public static void main(String args[]){
4. Employee e=new Employee();//object is created
5. e.setName("Arjun");//setting value to the object
6. System.out.println(e.getName());
7. }}

## JavaBean Properties

A JavaBean property is a named feature that can be accessed by the user of the object. The feature can be of any Java data type, containing the classes that you define.

A JavaBean property may be read, write, read-only, or write-only. JavaBean features are accessed through two methods in the JavaBean's implementation class:

**1. getPropertyName ()**

For example, if the property name is firstName, the method name would be getFirstName() to read that property. This method is called the accessor.

**2. setPropertyName ()**

For example, if the property name is firstName, the method name would be setFirstName() to write that property. This method is called the mutator.

### Advantages of JavaBean

The following are the advantages of JavaBean:/p>

* The JavaBean properties and methods can be exposed to another application.
* It provides an easiness to reuse the software components.

### Disadvantages of JavaBean

The following are the disadvantages of JavaBean:

* JavaBeans are mutable. So, it can't take advantages of immutable objects.
* Creating the setter and getter method for each property separately may lead to the boilerplate code.

# jsp:useBean action tag

The jsp:useBean action tag is used to locate or instantiate a bean class. If bean object of the Bean class is already created, it doesn't create the bean depending on the scope. But if object of bean is not created, it instantiates the bean.

## Syntax of jsp:useBean action tag

1. <jsp:useBean id= "instanceName" scope= "page | request | session | application"
2. class= "packageName.className" type= "packageName.className"
3. beanName="packageName.className | <%= expression >" >
4. </jsp:useBean>

### Attributes and Usage of jsp:useBean action tag

1. **id:** is used to identify the bean in the specified scope.
2. **scope:** represents the scope of the bean. It may be page, request, session or application. The default scope is page.
   * **page:** specifies that you can use this bean within the JSP page. The default scope is page.
   * **request:** specifies that you can use this bean from any JSP page that processes the same request. It has wider scope than page.
   * **session:** specifies that you can use this bean from any JSP page in the same session whether processes the same request or not. It has wider scope than request.
   * **application:** specifies that you can use this bean from any JSP page in the same application. It has wider scope than session.
3. **class:** instantiates the specified bean class (i.e. creates an object of the bean class) but it must have no-arg or no constructor and must not be abstract.
4. **type:** provides the bean a data type if the bean already exists in the scope. It is mainly used with class or beanName attribute. If you use it without class or beanName, no bean is instantiated.
5. **beanName:** instantiates the bean using the java.beans.Beans.instantiate() method.

### Simple example of jsp:useBean action tag

In this example, we are simply invoking the method of the Bean class.

#### For the example of setProperty, getProperty and useBean tags, visit next page.

### Calculator.java (a simple Bean class)

1. package com.javatpoint;
2. public class Calculator{
4. public int cube(int n){return n\*n\*n;}
6. }

### index.jsp file

1. <jsp:useBean id="obj" class="com.javatpoint.Calculator"/>
3. <%
4. int m=obj.cube(5);
5. out.print("cube of 5 is "+m);

# JSP Implicit Objects

There are **9 jsp implicit objects**. These objects are *created by the web container* that are available to all the jsp pages.

The available implicit objects are out, request, config, session, application etc.

A list of the 9 implicit objects is given below:

|  |  |
| --- | --- |
| **Object** | **Type** |
| Out | JspWriter |
| Request | HttpServletRequest |
| response | HttpServletResponse |
| Config | ServletConfig |
| application | ServletContext |
| Session | HttpSession |
| pageContext | PageContext |
| Page | Object |
| exception | Throwable |

### 1) JSP out implicit object

For writing any data to the buffer, JSP provides an implicit object named out. It is the object of JspWriter. In case of servlet you need to write:

1. PrintWriter out=response.getWriter();

But in JSP, you don't need to write this code.

### Example of out implicit object

In this example we are simply displaying date and time.

### index.jsp

1. <html>
2. <body>
3. <% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
4. </body>
5. </html>



# 2) JSP request implicit object

The **JSP request** is an implicit object of type HttpServletRequest i.e. created for each jsp request by the web container. It can be used to get request information such as parameter, header information, remote address, server name, server port, content type, character encoding etc.

It can also be used to set, get and remove attributes from the jsp request scope.

Let's see the simple example of request implicit object where we are printing the name of the user with welcome message.

### Example of JSP request implicit object

### index.html

1. <form action="welcome.jsp">
2. <input type="text" name="uname">
3. <input type="submit" value="go"><br/>
4. </form>

### welcome.jsp

1. <%
2. String name=request.getParameter("uname");
3. out.print("welcome "+name);
4. %>

#### Outputjsp request implicit object output 2

#### 3) JSP response implicit object

In JSP, response is an implicit object of type HttpServletResponse. The instance of HttpServletResponse is created by the web container for each jsp request.

It can be used to add or manipulate response such as redirect response to another resource, send error etc.

**Example of response implicit object**

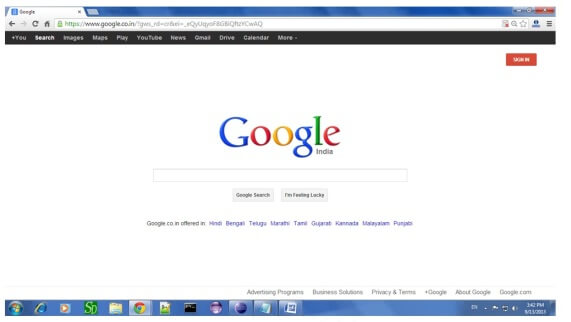
**index.html**

1. <form action="welcome.jsp">
2. <input type="text" name="uname">
3. <input type="submit" value="go"><br/>
4. </form>

**welcome.jsp**

1. <%
2. response.sendRedirect("http://www.google.com");
3. %>

**Output**



# 4) JSP config implicit object

In JSP, config is an implicit object of type *ServletConfig*. This object can be used to get initialization parameter for a particular JSP page. The config object is created by the web container for each jsp page.

Generally, it is used to get initialization parameter from the web.xml file.

### Example of config implicit object:

**index.html**

1. <form action="welcome">
2. <input type="text" name="uname">
3. <input type="submit" value="go"><br/>
4. </form>

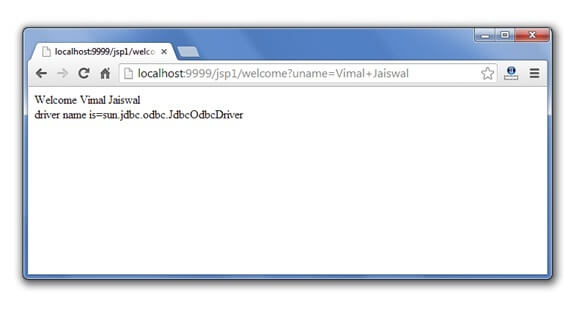
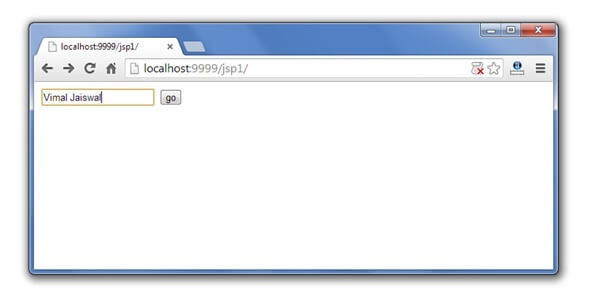
**web.xml file**

1. <web-app>
3. <servlet>
4. <servlet-name>sonoojaiswal</servlet-name>
5. <jsp-file>/welcome.jsp</jsp-file>
7. <init-param>
8. <param-name>dname</param-name>
9. <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
10. </init-param>
12. </servlet>
14. <servlet-mapping>
15. <servlet-name>sonoojaiswal</servlet-name>
16. <url-pattern>/welcome</url-pattern>
17. </servlet-mapping>
19. </web-app>

**welcome.jsp**

1. <%
2. out.print("Welcome "+request.getParameter("uname"));
4. String driver=config.getInitParameter("dname");
5. out.print("driver name is="+driver);
6. %>

#### Output



# ) JSP application implicit object

In JSP, application is an implicit object of type *ServletContext*.

The instance of ServletContext is created only once by the web container when application or project is deployed on the server.

This object can be used to get initialization parameter from configuaration file (web.xml). It can also be used to get, set or remove attribute from the application scope.

This initialization parameter can be used by all jsp pages.

Example of application implicit object:

**index.html**

1. <form action="welcome">
2. <input type="text" name="uname">
3. <input type="submit" value="go"><br/>
4. </form>

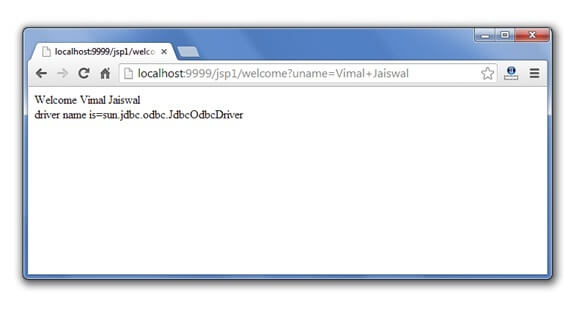
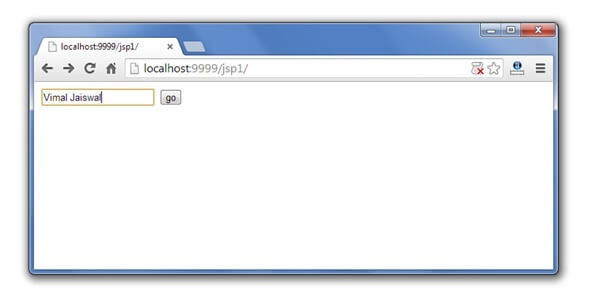
**web.xml file**

1. <web-app>
3. <servlet>
4. <servlet-name>sonoojaiswal</servlet-name>
5. <jsp-file>/welcome.jsp</jsp-file>
6. </servlet>
8. <servlet-mapping>
9. <servlet-name>sonoojaiswal</servlet-name>
10. <url-pattern>/welcome</url-pattern>
11. </servlet-mapping>
13. <context-param>
14. <param-name>dname</param-name>
15. <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
16. </context-param>
18. </web-app>

**welcome.jsp**

1. <%
3. out.print("Welcome "+request.getParameter("uname"));
5. String driver=application.getInitParameter("dname");
6. out.print("driver name is="+driver);
8. %>

#### Output



# 6) session implicit object

|  |
| --- |
| In JSP, session is an implicit object of type HttpSession.The Java developer can use this object to set,get or remove attribute or to get session information. |

# 7) pageContext implicit object

|  |
| --- |
| In JSP, pageContext is an implicit object of type PageContext class.The pageContext object can be used to set,get or remove attribute from one of the following scopes:   * page * request * session * application |
| In JSP, page scope is the default scope. |

### Example of pageContext implicit object

### index.html

1. <html>
2. <body>
3. <form action="welcome.jsp">
4. <input type="text" name="uname">
5. <input type="submit" value="go"><br/>
6. </form>
7. </body>
8. </html>

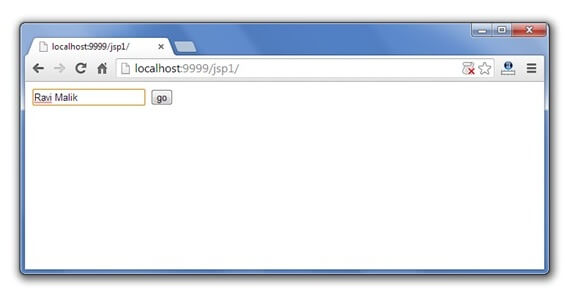
### welcome.jsp

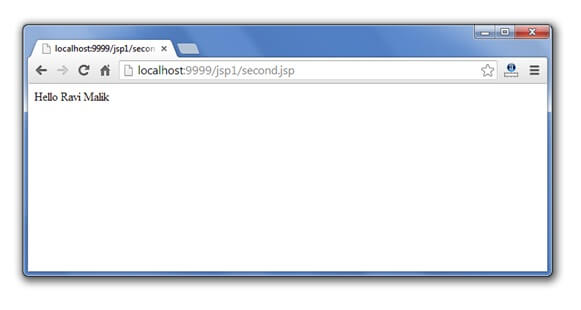
1. <html>
2. <body>
3. <%
5. String name=request.getParameter("uname");
6. out.print("Welcome "+name);
8. pageContext.setAttribute("user",name,PageContext.SESSION\_SCOPE);
10. <a href="second.jsp">second jsp page</a>
12. %>
13. </body>
14. </html>

### second.jsp

1. <html>
2. <body>
3. <%
5. String name=(String)pageContext.getAttribute("user",PageContext.SESSION\_SCOPE);
6. out.print("Hello "+name);
8. %>
9. </body>
10. </html>

#### Output





# 9) exception implicit object

|  |
| --- |
| In JSP, exception is an implicit object of type java.lang.Throwable class. This object can be used to print the exception. But it can only be used in error pages.It is better to learn it after page directive. Let's see a simple example: |

**Example of exception implicit object:**

**error.jsp**

1. <%@ page isErrorPage="true" %>
2. <html>
3. <body>
4. Sorry following exception occured:<%= exception %>
5. </body>
6. </html>

# JSP - Session Tracking

In this chapter, we will discuss session tracking in JSP. HTTP is a "stateless" protocol which means each time a client retrieves a Webpage, the client opens a separate connection to the Web server and the server automatically does not keep any record of previous client request.

## Maintaining Session Between Web Client And Server

Let us now discuss a few options to maintain the session between the Web Client and the Web Server −

### Cookies

A webserver can assign a unique session ID as a cookie to each web client and for subsequent requests from the client they can be recognized using the received cookie.

This may not be an effective way as the browser at times does not support a cookie. It is not recommended to use this procedure to maintain the sessions.

### Hidden Form Fields

A web server can send a hidden HTML form field along with a unique session ID as follows −

<input type = "hidden" name = "sessionid" value = "12345">

This entry means that, when the form is submitted, the specified name and value are automatically included in the **GET** or the **POST** data. Each time the web browser sends the request back, the **session\_id** value can be used to keep the track of different web browsers.

This can be an effective way of keeping track of the session but clicking on a regular (<A HREF...>) hypertext link does not result in a form submission, so hidden form fields also cannot support general session tracking.

### URL Rewriting

You can append some extra data at the end of each URL. This data identifies the session; the server can associate that session identifier with the data it has stored about that session.

For example, with **http://tutorialspoint.com/file.htm;sessionid=12345**, the session identifier is attached as **sessionid = 12345** which can be accessed at the web server to identify the client.

URL rewriting is a better way to maintain sessions and works for the browsers when they don't support cookies. The drawback here is that you will have to generate every URL dynamically to assign a session ID though page is a simple static HTML page.

## The session Object

Apart from the above mentioned options, JSP makes use of the servlet provided HttpSession Interface. This interface provides a way to identify a user across.

* a one page request or
* visit to a website or
* store information about that user

By default, JSPs have session tracking enabled and a new HttpSession object is instantiated for each new client automatically. Disabling session tracking requires explicitly turning it off by setting the page directive session attribute to false as follows −

<%@ page session = "false" %>

The JSP engine exposes the HttpSession object to the JSP author through the implicit **session** object. Since **session** object is already provided to the JSP programmer, the programmer can immediately begin storing and retrieving data from the object without any initialization or **getSession()**.

Here is a summary of important methods available through the session object −

|  |  |
| --- | --- |
| **S.No.** | **Method & Description** |
| 1 | **public Object getAttribute(String name)**  This method returns the object bound with the specified name in this session, or null if no object is bound under the name. |
| 2 | **public Enumeration getAttributeNames()**  This method returns an Enumeration of String objects containing the names of all the objects bound to this session. |
| 3 | **public long getCreationTime()**  This method returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT. |
| 4 | **public String getId()**  This method returns a string containing the unique identifier assigned to this session. |
| 5 | **public long getLastAccessedTime()**  This method returns the last time the client sent a request associated with the this session, as the number of milliseconds since midnight January 1, 1970 GMT. |
| 6 | **public int getMaxInactiveInterval()**  This method returns the maximum time interval, in seconds, that the servlet container will keep this session open between client accesses. |
| 7 | **public void invalidate()**  This method invalidates this session and unbinds any objects bound to it. |
| 8 | **public boolean isNew()**  This method returns true if the client does not yet know about the session or if the client chooses not to join the session. |
| 9 | **public void removeAttribute(String name)**  This method removes the object bound with the specified name from this session. |
| 10 | **public void setAttribute(String name, Object value)**  This method binds an object to this session, using the name specified. |
| 11 | **public void setMaxInactiveInterval(int interval)**  This method specifies the time, in seconds, between client requests before the servlet container will invalidate this session. |

### Session Tracking Example

This example describes how to use the HttpSession object to find out the creation time and the last-accessed time for a session. We would associate a new session with the request if one does not already exist.

<%@ page import = "java.io.\*,java.util.\*" %>

<%

// Get session creation time.

Date createTime = new Date(session.getCreationTime());

// Get last access time of this Webpage.

Date lastAccessTime = new Date(session.getLastAccessedTime());

String title = "Welcome Back to my website";

Integer visitCount = new Integer(0);

String visitCountKey = new String("visitCount");

String userIDKey = new String("userID");

String userID = new String("ABCD");

// Check if this is new comer on your Webpage.

if (session.isNew() ){

title = "Welcome to my website";

session.setAttribute(userIDKey, userID);

session.setAttribute(visitCountKey, visitCount);

}

visitCount = (Integer)session.getAttribute(visitCountKey);

visitCount = visitCount + 1;

userID = (String)session.getAttribute(userIDKey);

session.setAttribute(visitCountKey, visitCount);

%>

<html>

<head>

<title>Session Tracking</title>

</head>

<body>

<center>

<h1>Session Tracking</h1>

</center>

<table border = "1" align = "center">

<tr bgcolor = "#949494">

<th>Session info</th>

<th>Value</th>

</tr>

<tr>

<td>id</td>

<td><% out.print( session.getId()); %></td>

</tr>

<tr>

<td>Creation Time</td>

<td><% out.print(createTime); %></td>

</tr>

<tr>

<td>Time of Last Access</td>

<td><% out.print(lastAccessTime); %></td>

</tr>

<tr>

<td>User ID</td>

<td><% out.print(userID); %></td>

</tr>

<tr>

<td>Number of visits</td>

<td><% out.print(visitCount); %></td>

</tr>

</table>

</body>

</html>

Now put the above code in **main.jsp** and try to access ***http://localhost:8080/main.jsp***. Once you run the URL, you will receive the following result −

### Welcome to my website

**Session Information**

|  |  |
| --- | --- |
| **Session info** | **value** |
| id | 0AE3EC93FF44E3C525B4351B77ABB2D5 |
| Creation Time | Tue Jun 08 17:26:40 GMT+04:00 2010 |
| Time of Last Access | Tue Jun 08 17:26:40 GMT+04:00 2010 |
| User ID | ABCD |
| Number of visits | 0 |

Now try to run the same JSP for the second time, you will receive the following result.

### Welcome Back to my website

**Session Information**

|  |  |
| --- | --- |
| **info type** | **value** |
| id | 0AE3EC93FF44E3C525B4351B77ABB2D5 |
| Creation Time | Tue Jun 08 17:26:40 GMT+04:00 2010 |
| Time of Last Access | Tue Jun 08 17:26:40 GMT+04:00 2010 |
| User ID | ABCD |
| Number of visits | 1 |

## Deleting Session Data

When you are done with a user's session data, you have several options −

* **Remove a particular attribute** − You can call the ***public void removeAttribute(String name)*** method to delete the value associated with the a particular key.
* **Delete the whole session** − You can call the ***public void invalidate()*** method to discard an entire session.
* **Setting Session timeout** − You can call the ***public void setMaxInactiveInterval(int interval)*** method to set the timeout for a session individually.
* **Log the user out** − The servers that support servlets 2.4, you can call **logout** to log the client out of the Web server and invalidate all sessions belonging to all the users.
* **web.xml Configuration** − If you are using Tomcat, apart from the above mentioned methods, you can configure the session time out in web.xml file as follows.

<session-config>

<session-timeout>15</session-timeout>

</session-config>

The timeout is expressed as minutes, and overrides the default timeout which is 30 minutes in Tomcat.

The **getMaxInactiveInterval( )** method in a servlet returns the timeout period for that session in seconds. So if your session is configured in web.xml for 15 minutes, **getMaxInactiveInterval( )** returns 900.

# JSP – Error Handling

Exception Handling is **the process to handle the runtime errors**. There may occur exception any time in your web application. So handling exceptions is a safer side for the web developer. In JSP, there are two ways to perform exception handling

we will discuss how to handle exceptions in JSP. When you are writing a JSP code, you might make coding errors which can occur at any part of the code. There may occur the following type of errors in your JSP code −

### Checked exceptions

A checked exception is an exception that is typically a user error or a problem that cannot be foreseen by the programmer. For example, if a file is to be opened, but the file cannot be found, an exception occurs. These exceptions cannot simply be ignored at the time of compilation.

### Runtime exceptions

A runtime exception is an exception that probably could have been avoided by the programmer. As opposed to the checked exceptions, runtime exceptions are ignored at the time of compliation.

### Errors

These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

We will further discuss ways to handle run time exception/error occuring in your JSP code.

## Using Exception Object

The exception object is an instance of a subclass of Throwable (e.g., java.lang. NullPointerException) and is only available in error pages. Following table lists out the important methods available in the Throwable class.

|  |  |
| --- | --- |
| **S.No.** | **Methods & Description** |
| 1 | **public String getMessage()**  Returns a detailed message about the exception that has occurred. This message is initialized in the Throwable constructor. |
| 2 | **public Throwable getCause()**  Returns the cause of the exception as represented by a Throwable object. |
| 3 | **public String toString()**  Returns the name of the class concatenated with the result of **getMessage()**. |
| 4 | **public void printStackTrace()**  Prints the result of **toString()** along with the stack trace to **System.err**, the error output stream. |
| 5 | **public StackTraceElement [] getStackTrace()**  Returns an array containing each element on the stack trace. The element at index 0 represents the top of the call stack, and the last element in the array represents the method at the bottom of the call stack. |
| 6 | **public Throwable fillInStackTrace()**  Fills the stack trace of this Throwable object with the current stack trace, adding to any previous information in the stack trace. |

JSP gives you an option to specify **Error Page** for each JSP. Whenever the page throws an exception, the JSP container automatically invokes the error page.

Following is an example to specifiy an error page for a **main.jsp**. To set up an error page, use the **<%@ page errorPage = "xxx" %>** directive.

<%@ page errorPage = "ShowError.jsp" %>

<html>

<head>

<title>Error Handling Example</title>

</head>

<body>

<%

// Throw an exception to invoke the error page

int x = 1;

if (x == 1) {

throw new RuntimeException("Error condition!!!");

}

%>

</body>

</html>

We will now write one Error Handling JSP ShowError.jsp, which is given below. Notice that the error-handling page includes the directive **<%@ page isErrorPage = "true" %>**. This directive causes the JSP compiler to generate the exception instance variable.

<%@ page isErrorPage = "true" %>

<html>

<head>

<title>Show Error Page</title>

</head>

<body>

<h1>Opps...</h1>

<p>Sorry, an error occurred.</p>

<p>Here is the exception stack trace: </p>

<pre><% exception.printStackTrace(response.getWriter()); %></pre>

</body>

</html>

Access the **main.jsp**, you will receive an output somewhat like the following −

java.lang.RuntimeException: Error condition!!!

......

Opps...

Sorry, an error occurred.

Here is the exception stack trace:

−

# Opps...

|  |  |
| --- | --- |
| **Error:** | java.lang.RuntimeException: Error condition!!! |
| **URI:** | /main.jsp |
| **Status code:** | 500 |
| **Stack trace:** | org.apache.jsp.main\_jsp.\_jspService(main\_jsp.java:65)  org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:68)  javax.servlet.http.HttpServlet.service(HttpServlet.java:722)  org.apache.jasper.servlet.JspServlet.service(JspServlet.java:265)  javax.servlet.http.HttpServlet.service(HttpServlet.java:722) |

**ASP.NET**

ASP.NET is a web framework designed and developed by Microsoft. It is used to develop websites, web applications and web services. It provides fantastic integration of HTML, CSS and JavaScript. It was first released in January 2002. It is built on the Common Language Runtime (CLR) and allows programmers to write code using any supported .NET language.

Our ASP.NET Tutorial includes all topics of ASP.NET Tutorial such as ASP.Net introduction, features, project, example, server controls, labels, textbox, button, hyperlink, radiobutton, calender, checkbox, fileupload, events handling, authentication, webforms model binding, html server control, compare validdator, range validator, validation summary, mvc introduction, mvc project, view, validation, entity framework, authentication etc..

**What is a Page Directive?**  
  
Basically, Page Directives are commands. These commands are used by the compiler when the page is compiled.  
  
**How to use the directives in an ASP.NET page**  
  
It is not difficult to add a directive to an ASP.NET page. It is simple to add directives to an ASP.NET page. You can write directives in the following format:  
  
<%@[Directive][Attributes]%>  
  
See the directive format, it starts with "<%@" and ends with "%>". The best way is to put the directive at the top of your page. But you can put a directive anywhere in a page. One more thing, you can put more than one attribute in a single directive.  
  
Here is the full list of directives:

* @Page
* @Master
* @Control
* @Import
* @Implements
* @Register
* @Assembly
* @MasterType
* @Output Cache
* @PreviousPageType
* @Reference

Let's discuss something about each directive.  
  
**@Page**  
  
When you want to specify the attributes for an ASP.NET page then you need to use @Page Directive. As you know, an ASP.NET page is a very important part of ASP.NET, so this directive is commonly used in ASP.NET.  
  
Example:

1. <%@Page Language="C#" AutoEventWIreup="false" CodeFile="Default.aspx.cs" Inherits="\_Default"%>

**@Master**  
  
Now you have some information about @Page Directives. The @Master Directive is quite similar to the @Page Directive. The only difference is that the @master directive is for Master pages. You need to note that, while using the @Master Directive you define the template page's property. Then any content page can inherit all the properties defined in the Master Page. But there are some properties that are only available in a Master Page.  
  
Example

1. <%@Master Language="C#" AutoEventWIreup="false" CodeFile="MasterPage1.master.cs" Inherits="MasterPage"%>

**@Control**  
@Control builds ASP.NET user controls. When you use the directive you define the properties to be inherited by the user controls and theses values are assigned to the user controls  
  
Example:

1. <%@Control Language="C#" Explicit="True" CodeFile="WebUserControl.ascx.cs" Inherits="WebUserControl" %>

**@Import**   
  
As you know you need to define namespaces in your .cs class before using a C# or VB class. So the @Import Directive imports namespaces. This directive supports just a single attribute "namespace" and this attribute takes a string value that specifies the namespace to be imported. One thing you need to note is that the @Import Directive cannot contain more than one attribute/value pair. But you can use multiple lines.  
  
Example:

1. <%@Import Namespace="System.Data"%>

**@Implements**   
  
The @Implements Directive gets the ASP.NET pages to implement .Net framework interfaces. This directive only supports a single attribute interface.  
  
Example:

1. <%@Implements Interface="System.Web.UI.IValidator"%>

**@Register**   
  
When you create a user control and you drag that user control onto your page then you will see the @Register directive. This directive registers your user control on the page so that the control can be accessed by the page.  
  
Example:

1. <%@ Register TagPrefix="MayTag Namespace="MyName.MyNameSpace" Assembly="MyAssembly"%>

**@Assembly**   
  
The @Assembly Directive attaches assemblies to the page or an ASP.NET user control thereby all the assembly classes and interfaces are available to the class. This directive supports the two attributes Name and src. The Name attribute defines the assembly name and the src attribute defines the source of the assembly.  
  
Example:

1. <%@Assembly Name="MyAssembly"%>
2. <%@Assembly src="MYAssembly.cs">

**@MasterType**   
  
The @MasterType Directive connects a class name to the ASP.NET page for getting strongly typed references or members contained in the specified Master Page. This directive supports the two attributes Typename and virtualpath. Typename sets the name of the derived class from which to get the strongly typed or reference members and virtualpath sets the location of the page from which these are retrieved.  
  
Example:

1. <%@MasterType VirtualPath="/MasterPage1.master"%>

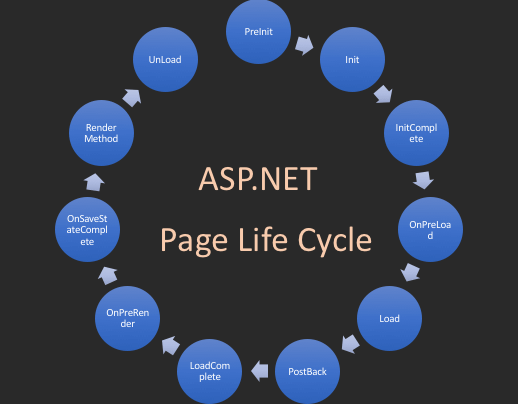
**@output cache**  
  
It controls the output caching policies of an ASP.NET page.  
  
Example:

1. <%@ OutputCache Duration ="180" VaryByParam="None"%>

**@Previouspagetype**  
  
This directive specifies the page from which any cross-page posting originates.  
  
**@Reference**  
  
This directive declares that another page or user control shout be complied along with the active page or control. This directive supports the single attribute virtualpath. It sets the location of the page or user control from which the active page will be referenced.  
  
Example:

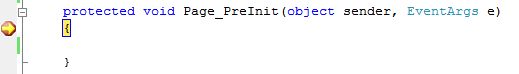
1. <%@Reference VirtualPayh="~/MyControl.ascx"%>

ASP.NET page life cycle. When an ASP.NET page runs, the page goes through a life cycle in which it performs a series of processing steps. These include initialization, instantiating controls, restoring and maintaining state, running event handler code, and rendering. The following are the various stages or events of ASP.Net page life cycle.



## PreInit

1. Check the IsPostBack property to determine whether this is the first time the page is being processed.
2. Create or re-create dynamic controls.
3. Set a master page dynamically.
4. Set the Theme property dynamically.

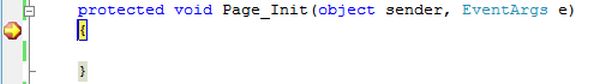


**Note**

If the request is a postback then the values of the controls have not yet been restored from the view state. If you set a control property at this stage, its value might be overwritten in the next event.

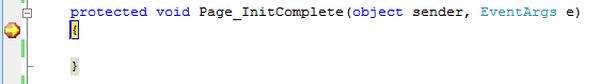
## Init

1. This event fires after each control has been initialized.
2. Each control's UniqueID is set and any skin settings have been applied.
3. Use this event to read or initialize control properties.
4. The "Init" event is fired first for the bottom-most control in the hierarchy, and then fired up the hierarchy until it is fired for the page itself.



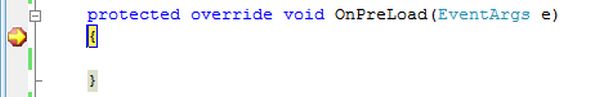
## InitComplete

1. Until now the viewstate values are not yet loaded, hence you can use this event to make changes to the view state that you want to ensure are persisted after the next postback.
2. Raised by the Page object.
3. Use this event for processing tasks that require all initialization to be complete.



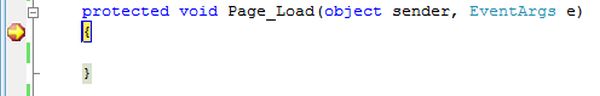
## OnPreLoad

1. Raised after the page loads view state for itself and all controls, and after it processes postback data that is included with the Request instance.
2. Before the Page instance raises this event, it loads view state for itself and all controls, and then processes any postback data included with the Request instance.
3. Loads ViewState: ViewState data are loaded to controls.
4. Loads Postback data: Postback data are now handed to the page controls.



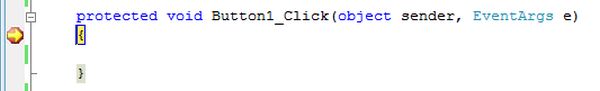
## Load

1. The Page object calls the OnLoad method on the Page object, and then recursively does the same for each child control until the page and all controls are loaded. The Load event of individual controls occurs after the Load event of the page.
2. This is the first place in the page lifecycle that all values are restored.
3. Most code checks the value of IsPostBack to avoid unnecessarily resetting state.
4. You may also call Validate and check the value of IsValid in this method.
5. You can also create dynamic controls in this method.
6. Use the OnLoad event method to set properties in controls and establish database connections.



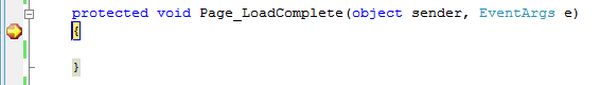
## Control PostBack Event(s)

1. ASP.NET now calls any events on the page or its controls that caused the PostBack to occur.
2. Use these events to handle specific control events, such as a Button control's Click event or a TextBox control's TextChanged event.
3. In a postback request, if the page contains validator controls, check the IsValid property of the Page and of individual validation controls before performing any processing.
4. This is just an example of a control event. Here it is the button click event that caused the postback.



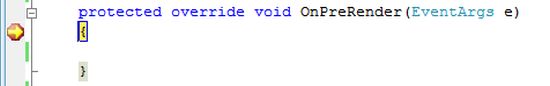
## LoadComplete

1. Raised at the end of the event-handling stage.
2. Use this event for tasks that require that all other controls on the page be loaded.



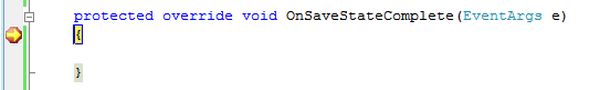
## OnPreRender

1. Raised after the Page object has created all controls that are required in order to render the page, including child controls of composite controls.
2. The Page object raises the PreRender event on the Page object, and then recursively does the same for each child control. The PreRender event of individual controls occurs after the PreRender event of the page.
3. The PreRender event of individual controls occurs after the PreRender event of the page.
4. Allows final changes to the page or its control.
5. This event takes place before saving ViewState, so any changes made here are saved.
6. For example: After this event, you cannot change any property of a button or change any viewstate value.
7. Each data bound control whose DataSourceID property is set calls its DataBind method.
8. Use the event to make final changes to the contents of the page or its controls.



## OnSaveStateComplete

1. Raised after view state and control state have been saved for the page and for all controls.
2. Before this event occurs, ViewState has been saved for the page and for all controls.
3. Any changes to the page or controls at this point will be ignored.
4. Use this event perform tasks that require the view state to be saved, but that do not make any changes to controls.

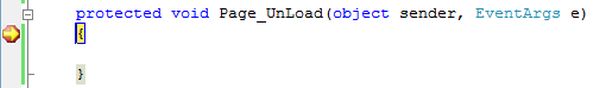


## Render Method

1. This is a method of the page object and its controls (and not an event).
2. The Render method generates the client-side HTML, Dynamic Hypertext Markup Language (DHTML), and script that are necessary to properly display a control at the browser.

## UnLoad

1. This event is used for cleanup code.
2. At this point, all processing has occurred and it is safe to dispose of any remaining objects, including the Page object.
3. Cleanup can be performed on:
   * Instances of classes, in other words objects
   * Closing opened files
   * Closing database connections.
4. This event occurs for each control and then for the page.
5. During the unload stage, the page and its controls have been rendered, so you cannot make further changes to the response stream.
6. If you attempt to call a method such as the Response.Write method then the page will throw an exception.



# What is Cross Page Posting in ASP.NET

By default, buttons have a postback property. When you click the button it reloads the page itself. However we can use the property PostBackUrl to redirect to another page. If you want to use the data of one page to another page without using session, object, or anything else, you can just use cross-page in your project.

Cross page posting means you are posting form data to another page. This is useful when you want to post data to another page and do not want incur the overhead of reloading the current page. The below code is given with a simple example. For this example we have to require two pages.

**Server controls in ASP.NET**

Controls are small building blocks of the graphical user interface, which include text boxes, buttons, check boxes, list boxes, labels, and numerous other tools. Using these tools, the users can enter data, make selections and indicate their preferences.

Controls are also used for structural jobs, like validation, data access, security, creating master pages, and data manipulation.

ASP.NET uses five types of web controls, which are:

* HTML controls
* HTML Server controls
* ASP.NET Server controls
* ASP.NET Ajax Server controls
* User controls and custom controls

ASP.NET server controls are the primary controls used in ASP.NET. These controls can be grouped into the following categories:

* **Validation controls** - These are used to validate user input and they work by running client-side script.
* **Data source controls** - These controls provides data binding to different data sources.
* **Data view controls** - These are various lists and tables, which can bind to data from data sources for displaying.
* **Personalization controls** - These are used for personalization of a page according to the user preferences, based on user information.
* **Login and security controls** - These controls provide user authentication.
* **Master pages** - These controls provide consistent layout and interface throughout the application.
* **Navigation controls** - These controls help in navigation. For example, menus, tree view etc.
* **Rich controls** - These controls implement special features. For example, AdRotator, FileUpload, and Calendar control.

The syntax for using server controls is:

<asp:controlType ID ="ControlID" runat="server" Property1=value1 [Property2=value2] />

In addition, visual studio has the following features, to help produce in error-free coding:

* Dragging and dropping of controls in design view
* IntelliSense feature that displays and auto-completes the properties
* The properties window to set the property values directly

# ASP.NET HTML Server Controls

HTML server controls are HTML elements that contain attributes to accessible at server side. By default, HTML elements on an ASP.NET Web page are not available to the server. These components are treated as simple text and pass through to the browser. We can convert an HTML element to server control by adding a **runat="server"** and an **id** attribute to the component.

Now, we can easily access it at code behind.

**Example**

1. <input id="UserName" type="text" size="50"runat="server" />

All the HTML Server controls can be accessed through the **Request** object.

## HTML Components

The following table contains commonly used HTML components.

|  |  |
| --- | --- |
| **Controls Name** | **Description** |
| Button | It is used to create HTML button. |
| Reset Button | It is used to reset all HTML form elements. |
| Submit Button | It is used to submit form data to the server. |
| Text Field | It is used to create text input. |
| Text Area | It is used to create a text area in the html form. |
| File | It is used to create a input type = "file" component which is used to upload file to the server. |
| Password | It is a password field which is used to get password from the user. |
| CheckBox | It creates a check box that user can select or clear. |
| Radio Button | A radio field which is used to get user choice. |
| Table | It allows us to present information in a tabular format. |
| Image | It displays an image on an HTML form |
| ListBox | It displays a list of items to the user. You can set the size from two or more to specify how many items you wish to show. |
| Dropdown | It displays a list of items to the user in a dropdown list. |
| Horizontal Rule | It displays a horizontal line across the HTML page. |

## Example

Here, we are implementing an HTML server control in the form.

**// htmlcontrolsexample.aspx**

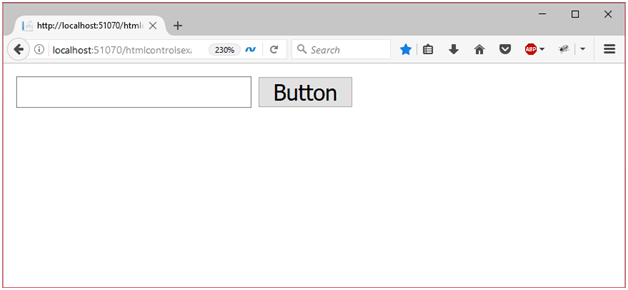
1. <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="htmlcontrolsexample.aspx.cs"
2. Inherits="asp.netexample.htmlcontrolsexample" %>
3. <!DOCTYPE html>
4. <html xmlns="http://www.w3.org/1999/xhtml">
5. <head runat="server">
6. <title></title>
7. </head>
8. <body>
9. <form id="form1" runat="server">
10. <div>
11. <input id="Text1" type="text" runat="server"/>
12. <asp:Button ID="Button1" runat="server" Text="Button" OnClick="Button1\_Click"/>
13. </div>
14. </form>
15. </body>
16. </html>

This application contains a code behind file.

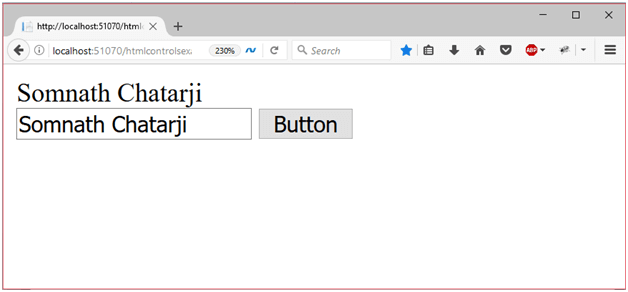
**// htmlcontrolsexample.aspx.cs**

1. using System;
2. namespace asp.netexample
3. {
4. public partial class htmlcontrolsexample : System.Web.UI.Page
5. {
6. protected void Page\_Load(object sender, EventArgs e)
7. {
8. }
9. protected void Button1\_Click(object sender, EventArgs e)
10. {
11. string a = Request.Form["Text1"];
12. Response.Write(a);
13. }
14. }
15. }

Output:



When we click the button after entering text, it responses back to client.



**Validation control in ASP.NET**

ASP.NET validation controls validate the user input data to ensure that useless, unauthenticated, or contradictory data don't get stored.

ASP.NET provides the following validation controls:

* RequiredFieldValidator
* RangeValidator
* CompareValidator
* RegularExpressionValidator
* CustomValidator
* ValidationSummary

## BaseValidator Class

The validation control classes are inherited from the BaseValidator class hence they inherit its properties and methods. Therefore, it would help to take a look at the properties and the methods of this base class, which are common for all the validation controls:

|  |  |
| --- | --- |
| **Members** | **Description** |
| ControlToValidate | Indicates the input control to validate. |
| Display | Indicates how the error message is shown. |
| EnableClientScript | Indicates whether client side validation will take. |
| Enabled | Enables or disables the validator. |
| ErrorMessage | Indicates error string. |
| Text | Error text to be shown if validation fails. |
| IsValid | Indicates whether the value of the control is valid. |
| SetFocusOnError | It indicates whether in case of an invalid control, the focus should switch to the related input control. |
| ValidationGroup | The logical group of multiple validators, where this control belongs. |
| Validate() | This method revalidates the control and updates the IsValid property. |

## RequiredFieldValidator Control

The RequiredFieldValidator control ensures that the required field is not empty. It is generally tied to a text box to force input into the text box.

The syntax of the control is as given:

<asp:RequiredFieldValidator ID="rfvcandidate"

runat="server" ControlToValidate ="ddlcandidate"

ErrorMessage="Please choose a candidate"

InitialValue="Please choose a candidate">

</asp:RequiredFieldValidator>

## RangeValidator Control

The RangeValidator control verifies that the input value falls within a predetermined range.

It has three specific properties:

|  |  |
| --- | --- |
| **Properties** | **Description** |
| Type | It defines the type of the data. The available values are: Currency, Date, Double, Integer, and String. |
| MinimumValue | It specifies the minimum value of the range. |
| MaximumValue | It specifies the maximum value of the range. |

The syntax of the control is as given:

<asp:RangeValidator ID="rvclass" runat="server" ControlToValidate="txtclass"

ErrorMessage="Enter your class (6 - 12)" MaximumValue="12"

MinimumValue="6" Type="Integer">

</asp:RangeValidator>

## CompareValidator Control

The CompareValidator control compares a value in one control with a fixed value or a value in another control.

It has the following specific properties:

|  |  |
| --- | --- |
| **Properties** | **Description** |
| Type | It specifies the data type. |
| ControlToCompare | It specifies the value of the input control to compare with. |
| ValueToCompare | It specifies the constant value to compare with. |
| Operator | It specifies the comparison operator, the available values are: Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual, and DataTypeCheck. |

The basic syntax of the control is as follows:

<asp:CompareValidator ID="CompareValidator1" runat="server"

ErrorMessage="CompareValidator">

</asp:CompareValidator>

## RegularExpressionValidator

The RegularExpressionValidator allows validating the input text by matching against a pattern of a regular expression. The regular expression is set in the ValidationExpression property.

The following table summarizes the commonly used syntax constructs for regular expressions:

|  |  |
| --- | --- |
| **Character Escapes** | **Description** |
| \b | Matches a backspace. |
| \t | Matches a tab. |
| \r | Matches a carriage return. |
| \v | Matches a vertical tab. |
| \f | Matches a form feed. |
| \n | Matches a new line. |
| \ | Escape character. |

Apart from single character match, a class of characters could be specified that can be matched, called the metacharacters.

|  |  |
| --- | --- |
| **Metacharacters** | **Description** |
| . | Matches any character except \n. |
| [abcd] | Matches any character in the set. |
| [^abcd] | Excludes any character in the set. |
| [2-7a-mA-M] | Matches any character specified in the range. |
| \w | Matches any alphanumeric character and underscore. |
| \W | Matches any non-word character. |
| \s | Matches whitespace characters like, space, tab, new line etc. |
| \S | Matches any non-whitespace character. |
| \d | Matches any decimal character. |
| \D | Matches any non-decimal character. |

Quantifiers could be added to specify number of times a character could appear.

|  |  |
| --- | --- |
| **Quantifier** | **Description** |
| \* | Zero or more matches. |
| + | One or more matches. |
| ? | Zero or one matches. |
| {N} | N matches. |
| {N,} | N or more matches. |
| {N,M} | Between N and M matches. |

The syntax of the control is as given:

<asp:RegularExpressionValidator ID="string" runat="server" ErrorMessage="string"

ValidationExpression="string" ValidationGroup="string">

</asp:RegularExpressionValidator>

## CustomValidator

The CustomValidator control allows writing application specific custom validation routines for both the client side and the server side validation.

The client side validation is accomplished through the ClientValidationFunction property. The client side validation routine should be written in a scripting language, such as JavaScript or VBScript, which the browser can understand.

The server side validation routine must be called from the control's ServerValidate event handler. The server side validation routine should be written in any .Net language, like C# or VB.Net.

The basic syntax for the control is as given:

<asp:CustomValidator ID="CustomValidator1" runat="server"

ClientValidationFunction=.cvf\_func. ErrorMessage="CustomValidator">

</asp:CustomValidator>

## ValidationSummary

The ValidationSummary control does not perform any validation but shows a summary of all errors in the page. The summary displays the values of the ErrorMessage property of all validation controls that failed validation.

The following two mutually inclusive properties list out the error message:

* **ShowSummary** : shows the error messages in specified format.
* **ShowMessageBox** : shows the error messages in a separate window.

The syntax for the control is as given:

<asp:ValidationSummary ID="ValidationSummary1" runat="server"

DisplayMode = "BulletList" ShowSummary = "true" HeaderText="Errors:" />

## Validation Groups

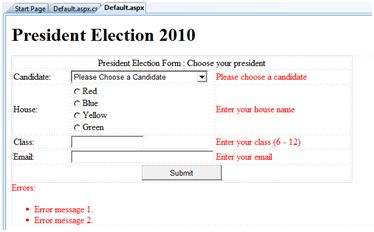
Complex pages have different groups of information provided in different panels. In such situation, a need might arise for performing validation separately for separate group. This kind of situation is handled using validation groups.

To create a validation group, you should put the input controls and the validation controls into the same logical group by setting their *ValidationGroup* property.

## Example

The following example describes a form to be filled up by all the students of a school, divided into four houses, for electing the school president. Here, we use the validation controls to validate the user input.

This is the form in design view:



The content file code is as given:

<form id="form1" runat="server">

<table style="width: 66%;">

<tr>

<td class="style1" colspan="3" align="center">

<asp:Label ID="lblmsg"

Text="President Election Form : Choose your president"

runat="server" />

</td>

</tr>

<tr>

<td class="style3">

Candidate:

</td>

<td class="style2">

<asp:DropDownList ID="ddlcandidate" runat="server" style="width:239px">

<asp:ListItem>Please Choose a Candidate</asp:ListItem>

<asp:ListItem>M H Kabir</asp:ListItem>

<asp:ListItem>Steve Taylor</asp:ListItem>

<asp:ListItem>John Abraham</asp:ListItem>

<asp:ListItem>Venus Williams</asp:ListItem>

</asp:DropDownList>

</td>

<td>

<asp:RequiredFieldValidator ID="rfvcandidate"

runat="server" ControlToValidate ="ddlcandidate"

ErrorMessage="Please choose a candidate"

InitialValue="Please choose a candidate">

</asp:RequiredFieldValidator>

</td>

</tr>

<tr>

<td class="style3">

House:

</td>

<td class="style2">

<asp:RadioButtonList ID="rblhouse" runat="server" RepeatLayout="Flow">

<asp:ListItem>Red</asp:ListItem>

<asp:ListItem>Blue</asp:ListItem>

<asp:ListItem>Yellow</asp:ListItem>

<asp:ListItem>Green</asp:ListItem>

</asp:RadioButtonList>

</td>

<td>

<asp:RequiredFieldValidator ID="rfvhouse" runat="server"

ControlToValidate="rblhouse" ErrorMessage="Enter your house name" >

</asp:RequiredFieldValidator>

<br />

</td>

</tr>

<tr>

<td class="style3">

Class:

</td>

<td class="style2">

<asp:TextBox ID="txtclass" runat="server"></asp:TextBox>

</td>

<td>

<asp:RangeValidator ID="rvclass"

runat="server" ControlToValidate="txtclass"

ErrorMessage="Enter your class (6 - 12)" MaximumValue="12"

MinimumValue="6" Type="Integer">

</asp:RangeValidator>

</td>

</tr>

<tr>

<td class="style3">

Email:

</td>

<td class="style2">

<asp:TextBox ID="txtemail" runat="server" style="width:250px">

</asp:TextBox>

</td>

<td>

<asp:RegularExpressionValidator ID="remail" runat="server"

ControlToValidate="txtemail" ErrorMessage="Enter your email"

ValidationExpression="\w+([-+.']\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*">

</asp:RegularExpressionValidator>

</td>

</tr>

<tr>

<td class="style3" align="center" colspan="3">

<asp:Button ID="btnsubmit" runat="server" onclick="btnsubmit\_Click"

style="text-align: center" Text="Submit" style="width:140px" />

</td>

</tr>

</table>

<asp:ValidationSummary ID="ValidationSummary1" runat="server"

DisplayMode ="BulletList" ShowSummary ="true" HeaderText="Errors:" />

</form>

The code behind the submit button:

protected void btnsubmit\_Click(object sender, EventArgs e)

{

if (Page.IsValid)

{

lblmsg.Text = "Thank You";

}

else

{

lblmsg.Text = "Fill up all the fields";

}

}

**PHP**

* PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.
* It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.
* PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the Unix side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time.
* PHP supports a large number of major protocols such as POP3, IMAP, and LDAP. PHP4 added support for Java and distributed object architectures (COM and CORBA), making n-tier development a possibility for the first time.
* PHP is forgiving: PHP language tries to be as forgiving as possible.
* PHP Syntax is C-Like.

## Common uses of PHP

* PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them.
* PHP can handle forms, i.e. gather data from files, save data to a file, through email you can send data, return data to the user.
* You add, delete, modify elements within your database through PHP.
* Access cookies variables and set cookies.
* Using PHP, you can restrict users to access some pages of your website.
* It can encrypt data.

## Characteristics of PHP

Five important characteristics make PHP's practical nature possible −

* Simplicity
* Efficiency
* Security
* Flexibility
* Familiarity

## "Hello World" Script in PHP

To get a feel for PHP, first start with simple PHP scripts. Since "Hello, World!" is an essential example, first we will create a friendly little "Hello, World!" script.

As mentioned earlier, PHP is embedded in HTML. That means that in amongst your normal HTML (or XHTML if you're cutting-edge) you'll have PHP statements like this −

<html>

<head>

<title>Hello World</title>

</head>

<body>

<?php echo "Hello, World!";?>

</body>

</html>

It will produce following result −

Hello, World!

# PHP - Syntax Overview

This chapter will give you an idea of very basic syntax of PHP and very important to make your PHP foundation strong.

## Escaping to PHP

The PHP parsing engine needs a way to differentiate PHP code from other elements in the page. The mechanism for doing so is known as 'escaping to PHP'. There are four ways to do this −

### Canonical PHP tags

The most universally effective PHP tag style is −

<?php...?>

If you use this style, you can be positive that your tags will always be correctly interpreted.

### Short-open (SGML-style) tags

Short or short-open tags look like this −

<?...?>

Short tags are, as one might expect, the shortest option You must do one of two things to enable PHP to recognize the tags −

* Choose the --enable-short-tags configuration option when you're building PHP.
* Set the short\_open\_tag setting in your php.ini file to on. This option must be disabled to parse XML with PHP because the same syntax is used for XML tags.

### ASP-style tags

ASP-style tags mimic the tags used by Active Server Pages to delineate code blocks. ASP-style tags look like this −

<%...%>

To use ASP-style tags, you will need to set the configuration option in your php.ini file.

### HTML script tags

HTML script tags look like this −

<script language = "PHP">...</script>

## Commenting PHP Code

A *comment* is the portion of a program that exists only for the human reader and stripped out before displaying the programs result. There are two commenting formats in PHP −

**Single-line comments** − They are generally used for short explanations or notes relevant to the local code. Here are the examples of single line comments.

<?

# This is a comment, and

# This is the second line of the comment

// This is a comment too. Each style comments only

print "An example with single line comments";

?>

**Multi-lines printing** − Here are the examples to print multiple lines in a single print statement −

<?

# First Example

print <<<END

This uses the "here document" syntax to output

multiple lines with $variable interpolation. Note

that the here document terminator must appear on a

line with just a semicolon no extra whitespace!

END;

# Second Example

print "This spans

multiple lines. The newlines will be

output as well";

?>

**Multi-lines comments** − They are generally used to provide pseudocode algorithms and more detailed explanations when necessary. The multiline style of commenting is the same as in C. Here are the example of multi lines comments.

<?

/\* This is a comment with multiline

Author : Mohammad Mohtashim

Purpose: Multiline Comments Demo

Subject: PHP

\*/

print "An example with multi line comments";

?>

## PHP is whitespace insensitive

Whitespace is the stuff you type that is typically invisible on the screen, including spaces, tabs, and carriage returns (end-of-line characters).

PHP whitespace insensitive means that it almost never matters how many whitespace characters you have in a row.one whitespace character is the same as many such characters.

For example, each of the following PHP statements that assigns the sum of 2 + 2 to the variable $four is equivalent −

$four = 2 + 2; // single spaces

$four <tab>=<tab2<tab>+<tab>2 ; // spaces and tabs

$four =

2+

2; // multiple lines

## PHP is case sensitive

Yeah it is true that PHP is a case sensitive language. Try out following example −

<html>

<body>

<?php

$capital = 67;

print("Variable capital is $capital<br>");

print("Variable CaPiTaL is $CaPiTaL<br>");

?>

</body>

</html>

This will produce the following result −

Variable capital is 67

Variable CaPiTaL is

## Statements are expressions terminated by semicolons

A *statement* in PHP is any expression that is followed by a semicolon (;).Any sequence of valid PHP statements that is enclosed by the PHP tags is a valid PHP program. Here is a typical statement in PHP, which in this case assigns a string of characters to a variable called $greeting −

$greeting = "Welcome to PHP!";

## Expressions are combinations of tokens

The smallest building blocks of PHP are the indivisible tokens, such as numbers (3.14159), strings (.two.), variables ($two), constants (TRUE), and the special words that make up the syntax of PHP itself like if, else, while, for and so forth

## Braces make blocks

Although statements cannot be combined like expressions, you can always put a sequence of statements anywhere a statement can go by enclosing them in a set of curly braces.

Here both statements are equivalent −

if (3 == 2 + 1)

print("Good - I haven't totally lost my mind.<br>");

if (3 == 2 + 1) {

print("Good - I haven't totally");

print("lost my mind.<br>");

}

## Running PHP Script from Command Prompt

Yes you can run your PHP script on your command prompt. Assuming you have following content in test.php file

<?php

echo "Hello PHP!!!!!";

?>

Now run this script as command prompt as follows −

$ php test.php

It will produce the following result −

Hello PHP!!!!!

**Variables**

The main way to store information in the middle of a PHP program is by using a variable.

Here are the most important things to know about variables in PHP.

* All variables in PHP are denoted with a leading dollar sign ($).
* The value of a variable is the value of its most recent assignment.
* Variables are assigned with the = operator, with the variable on the left-hand side and the expression to be evaluated on the right.
* Variables can, but do not need, to be declared before assignment.
* Variables in PHP do not have intrinsic types - a variable does not know in advance whether it will be used to store a number or a string of characters.
* Variables used before they are assigned have default values.
* PHP does a good job of automatically converting types from one to another when necessary.
* PHP variables are Perl-like.

PHP has a total of eight data types which we use to construct our variables −

* **Integers** − are whole numbers, without a decimal point, like 4195.
* **Doubles** − are floating-point numbers, like 3.14159 or 49.1.
* **Booleans** − have only two possible values either true or false.
* **NULL** − is a special type that only has one value: NULL.
* **Strings** − are sequences of characters, like 'PHP supports string operations.'
* **Arrays** − are named and indexed collections of other values.
* **Objects** − are instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.
* **Resources** − are special variables that hold references to resources external to PHP (such as database connections).

## What is the Form?

A Document that containing black fields, that the user can fill the data or user can select the data.Casually the data will store in the data base

### Example

Below example shows the form with some specific actions by using post method.

<html>

<head>

<title>PHP Form Validation</title>

</head>

<body>

<?php

// define variables and set to empty values

$name = $email = $gender = $comment = $website = "";

if ($\_SERVER["REQUEST\_METHOD"] == "POST") {

$name = test\_input($\_POST["name"]);

$email = test\_input($\_POST["email"]);

$website = test\_input($\_POST["website"]);

$comment = test\_input($\_POST["comment"]);

$gender = test\_input($\_POST["gender"]);

}

function test\_input($data) {

$data = trim($data);

$data = stripslashes($data);

$data = htmlspecialchars($data);

return $data;

}

?>

<h2>Tutorials Point Absolute classes registration</h2>

<form method = "post" action = "/php/php\_form\_introduction.htm">

<table>

<tr>

<td>Name:</td>

<td><input type = "text" name = "name"></td>

</tr>

<tr>

<td>E-mail:</td>

<td><input type = "text" name = "email"></td>

</tr>

<tr>

<td>Specific Time:</td>

<td><input type = "text" name = "website"></td>

</tr>

<tr>

<td>Class details:</td>

<td><textarea name = "comment" rows = "5" cols = "40"></textarea></td>

</tr>

<tr>

<td>Gender:</td>

<td>

<input type = "radio" name = "gender" value = "female">Female

<input type = "radio" name = "gender" value = "male">Male

</td>

</tr>

<tr>

<td>

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

</td>

</tr>

</table>

</form>

<?php

echo "<h2>Your Given details are as :</h2>";

echo $name;

echo "<br>";

echo $email;

echo "<br>";

echo $website;

echo "<br>";

echo $comment;

echo "<br>";

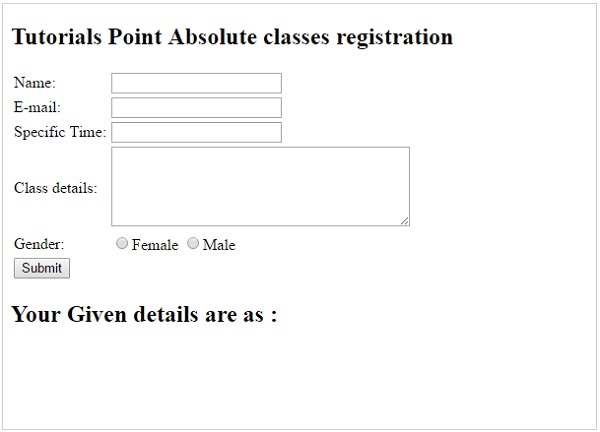
echo $gender;

?>

</body>

</html>

It will produce the following result −



# PHP - Sending Emails using PHP

PHP must be configured correctly in the **php.ini** file with the details of how your system sends email. Open php.ini file available in **/etc/** directory and find the section headed **[mail function]**.

Windows users should ensure that two directives are supplied. The first is called SMTP that defines your email server address. The second is called sendmail\_from which defines your own email address.

The configuration for Windows should look something like this −

[mail function]

; For Win32 only.

SMTP = smtp.secureserver.net

; For win32 only

sendmail\_from = webmaster@tutorialspoint.com

Linux users simply need to let PHP know the location of their **sendmail** application. The path and any desired switches should be specified to the sendmail\_path directive.

The configuration for Linux should look something like this −

[mail function]

; For Win32 only.

SMTP =

; For win32 only

sendmail\_from =

; For Unix only

sendmail\_path = /usr/sbin/sendmail -t -i

Now you are ready to go −

## Sending plain text email

PHP makes use of **mail()** function to send an email. This function requires three mandatory arguments that specify the recipient's email address, the subject of the the message and the actual message additionally there are other two optional parameters.

mail( to, subject, message, headers, parameters );

Here is the description for each parameters.

|  |  |
| --- | --- |
| **Sr.No** | **Parameter & Description** |
| 1 | **to**  Required. Specifies the receiver / receivers of the email |
| 2 | **subject**  Required. Specifies the subject of the email. This parameter cannot contain any newline characters |
| 3 | **message**  Required. Defines the message to be sent. Each line should be separated with a LF (\n). Lines should not exceed 70 characters |
| 4 | **headers**  Optional. Specifies additional headers, like From, Cc, and Bcc. The additional headers should be separated with a CRLF (\r\n) |
| 5 | **parameters**  Optional. Specifies an additional parameter to the send mail program |

As soon as the mail function is called PHP will attempt to send the email then it will return true if successful or false if it is failed.

Multiple recipients can be specified as the first argument to the mail() function in a comma separated list.

## Sending HTML email

When you send a text message using PHP then all the content will be treated as simple text. Even if you will include HTML tags in a text message, it will be displayed as simple text and HTML tags will not be formatted according to HTML syntax. But PHP provides option to send an HTML message as actual HTML message.

While sending an email message you can specify a Mime version, content type and character set to send an HTML email.

### Example

Following example will send an HTML email message to xyz@somedomain.com copying it to afgh@somedomain.com. You can code this program in such a way that it should receive all content from the user and then it should send an email.

<html>

<head>

<title>Sending HTML email using PHP</title>

</head>

<body>

<?php

$to = "xyz@somedomain.com";

$subject = "This is subject";

$message = "<b>This is HTML message.</b>";

$message .= "<h1>This is headline.</h1>";

$header = "From:abc@somedomain.com \r\n";

$header .= "Cc:afgh@somedomain.com \r\n";

$header .= "MIME-Version: 1.0\r\n";

$header .= "Content-type: text/html\r\n";

$retval = mail ($to,$subject,$message,$header);

if( $retval == true ) {

echo "Message sent successfully...";

}else {

echo "Message could not be sent...";

}

?>

</body>

</html>

## Sending attachments with email

To send an email with mixed content requires to set **Content-type** header to **multipart/mixed**. Then text and attachment sections can be specified within **boundaries**.

A boundary is started with two hyphens followed by a unique number which can not appear in the message part of the email. A PHP function **md5()** is used to create a 32 digit hexadecimal number to create unique number. A final boundary denoting the email's final section must also end with two hyphens.

<?php

// request variables // important

$from = $\_REQUEST["from"];

$emaila = $\_REQUEST["emaila"];

$filea = $\_REQUEST["filea"];

if ($filea) {

function mail\_attachment ($from , $to, $subject, $message, $attachment){

$fileatt = $attachment; // Path to the file

$fileatt\_type = "application/octet-stream"; // File Type

$start = strrpos($attachment, '/') == -1 ?

strrpos($attachment, '//') : strrpos($attachment, '/')+1;

$fileatt\_name = substr($attachment, $start,

strlen($attachment)); // Filename that will be used for the

file as the attachment

$email\_from = $from; // Who the email is from

$subject = "New Attachment Message";

$email\_subject = $subject; // The Subject of the email

$email\_txt = $message; // Message that the email has in it

$email\_to = $to; // Who the email is to

$headers = "From: ".$email\_from;

$file = fopen($fileatt,'rb');

$data = fread($file,filesize($fileatt));

fclose($file);

$msg\_txt="\n\n You have recieved a new attachment message from $from";

$semi\_rand = md5(time());

$mime\_boundary = "==Multipart\_Boundary\_x{$semi\_rand}x";

$headers .= "\nMIME-Version: 1.0\n" . "Content-Type: multipart/mixed;\n" . "

boundary=\"{$mime\_boundary}\"";

$email\_txt .= $msg\_txt;

$email\_message .= "This is a multi-part message in MIME format.\n\n" .

"--{$mime\_boundary}\n" . "Content-Type:text/html;

charset = \"iso-8859-1\"\n" . "Content-Transfer-Encoding: 7bit\n\n" .

$email\_txt . "\n\n";

$data = chunk\_split(base64\_encode($data));

$email\_message .= "--{$mime\_boundary}\n" . "Content-Type: {$fileatt\_type};\n" .

" name = \"{$fileatt\_name}\"\n" . //"Content-Disposition: attachment;\n" .

//" filename = \"{$fileatt\_name}\"\n" . "Content-Transfer-Encoding:

base64\n\n" . $data . "\n\n" . "--{$mime\_boundary}--\n";

$ok = mail($email\_to, $email\_subject, $email\_message, $headers);

if($ok) {

echo "File Sent Successfully.";

unlink($attachment); // delete a file after attachment sent.

}else {

die("Sorry but the email could not be sent. Please go back and try again!");

}

}

move\_uploaded\_file($\_FILES["filea"]["tmp\_name"],

'temp/'.basename($\_FILES['filea']['name']));

mail\_attachment("$from", "youremailaddress@gmail.com",

"subject", "message", ("temp/".$\_FILES["filea"]["name"]));

}

?>

<html>

<head>

<script language = "javascript" type = "text/javascript">

function CheckData45() {

with(document.filepost) {

if(filea.value ! = "") {

document.getElementById('one').innerText =

"Attaching File ... Please Wait";

}

}

}

</script>

</head>

<body>

<table width = "100%" height = "100%" border = "0"

cellpadding = "0" cellspacing = "0">

<tr>

<td align = "center">

<form name = "filepost" method = "post"

action = "file.php" enctype = "multipart/form-data" id = "file">

<table width = "300" border = "0" cellspacing = "0"

cellpadding = "0">

<tr valign = "bottom">

<td height = "20">Your Name:</td>

</tr>

<tr>

<td><input name = "from" type = "text"

id = "from" size = "30"></td>

</tr>

<tr valign = "bottom">

<td height = "20">Your Email Address:</td>

</tr>

<tr>

<td class = "frmtxt2"><input name = "emaila"

type = "text" id = "emaila" size = "30"></td>

</tr>

<tr>

<td height = "20" valign = "bottom">Attach File:</td>

</tr>

<tr valign = "bottom">

<td valign = "bottom"><input name = "filea"

type = "file" id = "filea" size = "16"></td>

</tr>

<tr>

<td height = "40" valign = "middle"><input

name = "Reset2" type = "reset" id = "Reset2" value = "Reset">

<input name = "Submit2" type = "submit"

value = "Submit" onClick = "return CheckData45()"></td>

</tr>

</table>

</form>

<center>

<table width = "400">

<tr>

<td id = "one">

</td>

</tr>

</table>

</center>

</td>

</tr>

</table>

</body>

# PHP - File Uploading

A PHP script can be used with a HTML form to allow users to upload files to the server. Initially files are uploaded into a temporary directory and then relocated to a target destination by a PHP script.

Information in the **phpinfo.php** page describes the temporary directory that is used for file uploads as **upload\_tmp\_dir** and the maximum permitted size of files that can be uploaded is stated as **upload\_max\_filesize**. These parameters are set into PHP configuration file **php.ini**

The process of uploading a file follows these steps −

* The user opens the page containing a HTML form featuring a text files, a browse button and a submit button.
* The user clicks the browse button and selects a file to upload from the local PC.
* The full path to the selected file appears in the text filed then the user clicks the submit button.
* The selected file is sent to the temporary directory on the server.
* The PHP script that was specified as the form handler in the form's action attribute checks that the file has arrived and then copies the file into an intended directory.
* The PHP script confirms the success to the user.

As usual when writing files it is necessary for both temporary and final locations to have permissions set that enable file writing. If either is set to be read-only then process will fail.

An uploaded file could be a text file or image file or any document.

## Creating an upload form

The following HTM code below creates an uploader form. This form is having method attribute set to **post** and enctype attribute is set to **multipart/form-data**

<?php

if(isset($\_FILES['image'])){

$errors= array();

$file\_name = $\_FILES['image']['name'];

$file\_size =$\_FILES['image']['size'];

$file\_tmp =$\_FILES['image']['tmp\_name'];

$file\_type=$\_FILES['image']['type'];

$file\_ext=strtolower(end(explode('.',$\_FILES['image']['name'])));

$extensions= array("jpeg","jpg","png");

if(in\_array($file\_ext,$extensions)=== false){

$errors[]="extension not allowed, please choose a JPEG or PNG file.";

}

if($file\_size > 2097152){

$errors[]='File size must be excately 2 MB';

}

if(empty($errors)==true){

move\_uploaded\_file($file\_tmp,"images/".$file\_name);

echo "Success";

}else{

print\_r($errors);

}

}

?>

<html>

<body>

<form action="" method="POST" enctype="multipart/form-data">

<input type="file" name="image" />

<input type="submit"/>

</form>

</body>

</html>

It will produce the following result −

Upload Form

## Creating an upload script

There is one global PHP variable called **$\_FILES**. This variable is an associate double dimension array and keeps all the information related to uploaded file. So if the value assigned to the input's name attribute in uploading form was **file**, then PHP would create following five variables −

* **$\_FILES['file']['tmp\_name']** − the uploaded file in the temporary directory on the web server.
* **$\_FILES['file']['name']** − the actual name of the uploaded file.
* **$\_FILES['file']['size']** − the size in bytes of the uploaded file.
* **$\_FILES['file']['type']** − the MIME type of the uploaded file.
* **$\_FILES['file']['error']** − the error code associated with this file upload.

### Example

Below example should allow upload images and gives back result as uploaded file information.

<?php

if(isset($\_FILES['image'])){

$errors= array();

$file\_name = $\_FILES['image']['name'];

$file\_size = $\_FILES['image']['size'];

$file\_tmp = $\_FILES['image']['tmp\_name'];

$file\_type = $\_FILES['image']['type'];

$file\_ext=strtolower(end(explode('.',$\_FILES['image']['name'])));

$extensions= array("jpeg","jpg","png");

if(in\_array($file\_ext,$extensions)=== false){

$errors[]="extension not allowed, please choose a JPEG or PNG file.";

}

if($file\_size > 2097152) {

$errors[]='File size must be excately 2 MB';

}

if(empty($errors)==true) {

move\_uploaded\_file($file\_tmp,"images/".$file\_name);

echo "Success";

}else{

print\_r($errors);

}

}

?>

<html>

<body>

<form action = "" method = "POST" enctype = "multipart/form-data">

<input type = "file" name = "image" />

<input type = "submit"/>

<ul>

<li>Sent file: <?php echo $\_FILES['image']['name']; ?>

<li>File size: <?php echo $\_FILES['image']['size']; ?>

<li>File type: <?php echo $\_FILES['image']['type'] ?>

</ul>

</form>

</body>

</html>

It will produce the following result −



A session creates a file in a temporary directory on the server where registered session variables and their values are stored. This data will be available to all pages on the site during that visit.

The location of the temporary file is determined by a setting in the **php.ini** file called **session.save\_path**. Before using any session variable make sure you have setup this path.

When a session is started following things happen −

* PHP first creates a unique identifier for that particular session which is a random string of 32 hexadecimal numbers such as 3c7foj34c3jj973hjkop2fc937e3443.
* A cookie called **PHPSESSID** is automatically sent to the user's computer to store unique session identification string.
* A file is automatically created on the server in the designated temporary directory and bears the name of the unique identifier prefixed by sess\_ ie sess\_3c7foj34c3jj973hjkop2fc937e3443.

When a PHP script wants to retrieve the value from a session variable, PHP automatically gets the unique session identifier string from the PHPSESSID cookie and then looks in its temporary directory for the file bearing that name and a validation can be done by comparing both values.

A session ends when the user loses the browser or after leaving the site, the server will terminate the session after a predetermined period of time, commonly 30 minutes duration.

## Starting a PHP Session

A PHP session is easily started by making a call to the **session\_start()** function.This function first checks if a session is already started and if none is started then it starts one. It is recommended to put the call to **session\_start()** at the beginning of the page.

Session variables are stored in associative array called **$\_SESSION[]**. These variables can be accessed during lifetime of a session.

The following example starts a session then register a variable called **counter** that is incremented each time the page is visited during the session.

Make use of **isset()** function to check if session variable is already set or not.

Put this code in a test.php file and load this file many times to see the result −

[Live Demo](http://tpcg.io/2Q0jAv)

<?php

session\_start();

if( isset( $\_SESSION['counter'] ) ) {

$\_SESSION['counter'] += 1;

}else {

$\_SESSION['counter'] = 1;

}

$msg = "You have visited this page ". $\_SESSION['counter'];

$msg .= "in this session.";

?>

<html>

<head>

<title>Setting up a PHP session</title>

</head>

<body>

<?php echo ( $msg ); ?>

</body>

</html>

It will produce the following result −

You have visited this page 1in this session.

## Destroying a PHP Session

A PHP session can be destroyed by **session\_destroy()** function. This function does not need any argument and a single call can destroy all the session variables. If you want to destroy a single session variable then you can use **unset()** function to unset a session variable.

Here is the example to unset a single variable −

<?php

unset($\_SESSION['counter']);

?>

Here is the call which will destroy all the session variables −

<?php

session\_destroy();

?>

**PHP Error exception**

PHP's Exception class implements the **Throwable** interface. **ErrorException** class extends the **Exception** class. ErrorException is meant to be explicitly thrown when you want to catch and handle errors that would otherwise be ignored, such as Notices or Warnings.

PHP core consists of following predefined error constants

| **Value** | **Constant** | **Description** |
| --- | --- | --- |
| 1 | E\_ERROR | Fatal run-time errors. |
| 2 | E\_WARNING | Run-time warnings (non-fatal errors). |
| 4 | E\_PARSE | Compile-time parse errors. |
| 8 | E\_NOTICE | Run-time notices. |
| 16 | E\_CORE\_ERROR | Fatal errors that occur during PHP's initial startup. |
| 32 | E\_CORE\_WARNING | Warnings (non-fatal errors) that occur during PHP's initial startup. |
| 64 | E\_COMPILE\_ERROR | Fatal compile-time errors. |
| 128 | E\_COMPILE\_WARNING | Compile-time warnings (non-fatal errors). |
| 256 | E\_USER\_ERROR | User-generated error message. |
| 512 | E\_USER\_WARNING | User-generated warning message. |
| 1024 | E\_USER\_NOTICE | User-generated notice message. |
| 2048 | E\_STRICT | If Enabled PHP suggests changes to your code to ensure interoperability and forward compatibility of your code. |
| 4096 | E\_RECOVERABLE\_ERROR | Catchable fatal error. |
| 8192 | E\_DEPRECATED | Run-time notices. |
| 16384 | E\_USER\_DEPRECATED | User-generated warning message. |
| 32767 | E\_ALL | All errors and warnings, E\_STRICT |

In addition to properties and methods inherited from Exception class, ErrorException class introduces one property and one method as follows −

protected int severity ;

final public getSeverity ( void ) : int

The severity of exception is represented by integer number associated with type of error in above table

## ErrorException example

In following script, a user defined function **errhandler** is set as Error handler with **set\_error\_handler()** function. It throws ErrorException when fatal error in event of file not found for reading is encountered.

## Example

<?php

function errhandler($severity, $message, $file, $line) {

   if (!(error\_reporting() & $severity)) {

      echo "no error";

      return;

   }

   throw new ErrorException("Fatal Error:No such file or directory", 0, E\_ERROR);

}

set\_error\_handler("errhandler");

/\* Trigger exception \*/

try{

   $data=file\_get\_contents("nofile.php");

   echo $data;

}

catch (ErrorException $e){

   echo $e->getMessage();

}

?>

Above example displays following output

## Output

Fatal Error:No such file or directory

**PHP ODBC**

Microsoft Open Database Connectivity (ODBC) interface is a C programming language interface that **makes it possible for applications to access data from a variety of database management systems (DBMSs)**. ODBC is a low-level, high-performance interface that is designed specifically for relational data stores.

The ODBC interface allows maximum interoperability-an application can access data in diverse DBMSs through a single interface. Moreover, that application will be independent of any DBMS from which it accesses data. Users of the application can add software components called drivers, which interface between an application and a specific DBMS.

## PHP Filter Introduction

This PHP filters is used to validate and filter data coming from insecure sources, like user input.

## Installation

From PHP 5.2.0, the filter functions are enabled by default. There is no installation needed to use these functions.

## PHP Filter Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| [filter\_has\_var()](https://www.w3schools.com/php/func_filter_has_var.asp) | Checks whether a variable of a specified input type exist |
| [filter\_id()](https://www.w3schools.com/php/func_filter_id.asp) | Returns the filter ID of a specified filter name |
| [filter\_input()](https://www.w3schools.com/php/func_filter_input.asp) | Gets an external variable (e.g. from form input) and optionally filters it |
| [filter\_input\_array()](https://www.w3schools.com/php/func_filter_input_array.asp) | Gets external variables (e.g. from form input) and optionally filters them |
| [filter\_list()](https://www.w3schools.com/php/func_filter_list.asp) | Returns a list of all supported filter names |
| [filter\_var()](https://www.w3schools.com/php/func_filter_var.asp) | Filters a variable with a specified filter |
| [filter\_var\_array()](https://www.w3schools.com/php/func_filter_var_array.asp) | Gets multiple variables and filter them |