Create HTML Forms

• A Tutorial •

by

William Bontrager

Programmer
and
Publisher
of

WillMaster Possibilities ezine

http://willmaster.com/possibilities/

mailto:possibilities@willmaster.com

A Peach-e-Book

Introduction

HTML forms are a means of collecting information. People fill in a form and/or select something. Then they click a button.

Forms don't actually process information.

For something to be done with the information, it must be sent somewhere. Information processing destinations can be CGI programs, JavaScript functions, mailto: links, or even a web page (which directs the browser to load the page at the specified URL).

This tutorial is about how to make forms and how to send the information, but not how to process the information after it has been sent off.

Forms begin with the <form> tag and end with the </form> tag. The <form> tag can contain several attributes.

Sometimes you'll need to determine the applicable attributes. Other times, installation directions for CGI programs or JavaScript functions specify the <form> tag attributes expected by the installation.

> **(Note: There is no need to copy the code in this article. A link where you can obtain the complete working form incorporating these examples, along with a script to process the form information, is in the last section.)**

Chapter 1 :: Where Information Is Sent

One <form> tag attribute specifies where the information will be sent. That's the "action" attribute.

~ Sending To CGI Programs

If the information is being sent to a CGI program on the same domain as the web page containing the form, the action attribute can contain either a relative URL or an absolute URL. With a relative URL, it would look something like this:

action="/cgi-bin/handler.cgi"

If the information is being sent to a CGI program on a different domain, the URL must be absolute (a complete http://... URL):

action="http://domain.com/cgi-bin/handler.cgi"

~ Sending To JavaScript Functions

When sending form information to a JavaScript function, the action attribute would look something like:

action="javascript:SomeFunction()"

or, the action attribute might be replaced with the JavaScript specific "onClick" attribute:

onClick="return SomeFunction()"

If the form information is to be sent to both a JavaScript function and to a CGI program, then both the action and the onClick attributes would be specified.

~ Sending To mailto: Links

To send form information to the visitor's email program, with the To: destination already filled in, use:

action="mailto:name@domain.com"
If you also want to specify a subject for the email:

action="mailto:name@domain.com?subject=My%20Subject"

(Use "%20" instead of spaces when specifying a subject.)

Note that not all browsers recognize the mailto: link. It is suggested you use this only as a last resort.

~ Sending To a Different Web Page

To send form information to a different web page so the browser opens the page at the new URL, use:

action="http://newdomain.com/page.html"

NOTE: Sending to a different web page requires method="GET" (see next section).

Chapter 2 :: How Information Is Sent

Another <form> tag attribute specifies how the information will be sent to wherever it's going. That's the "method" attribute. The method attribute is either

method="POST"

or

method="GET"

Which you use depends on how the destination program or function wants to receive the information.

~ Sending With Method GET

method="GET" is used if you want to send information somewhere via a browser URL. You've seen URLs that send information; they look something like:

http://domain.com/handler.cgi?color=red&shape=round

In the above URL, the part after the question mark is information sent to handler.cgi. Multiple information chunks are separated with an ampersand.

The GET method can send only a limited amount of information. The limitation depends on the server where the server where the information is sent to. The limitation can be as little as 256 bytes but is often 1k or more.

Another limitation of the GET method is that the information being sent is visible in the browser's address bar. In some cases this is of no consequence. In others, it is unacceptable.

Some CGI programs are written to accept information via the GET or the POST method, some to accept only the GET method.

~ Sending With Method POST

method=”POST” is the most common method used to send information from a form to an information processing program or function. This is the method used when sending form information to JavaScript functions. Most CGI programs are written to accept information with the POST method, some to accept only the POST method.

The POST method can send much more information than the typical GET method. Most browsers and servers limit the amount of POST information to about 32k.

With POST, the information is not sent via the URL. The sending is invisible to the site visitor.

It is not necessary to know the exact interior mechanism of how POST works, because all you have to do is specify the method and the browser decides how to accomplish your directive. A general overview: depending on its destination, the POST information could be transferred within the browser itself (if sent to a JavaScript function) or over the internet via a method known as “standard input” — the default or standard input channel expected by the destination (such as a server for a CGI program).

Chapter 3 :: Specifying the Encoding Method for the Information Being Sent

The final <form> tag attribute we'll cover in this tutorial specifies how the information will be encoded before sending it to wherever it's going. That's the "enctype" attribute.

The browser automatically encodes the information depending on the enctype attribute. Encoding transforms special non-alphanumeric characters that could otherwise wreak havoc during transmission or upon receipt into a series of characters that the receiving program or function can recognize. Encoding also inserts separators between the information chunks.

Often you don't have to concern yourself with the enctype. If you do not specify one, the default is assumed.

The default enctype attribute is:

enctype="application/x-www-form-urlencoded"

If your form includes file uploads, you will need

enctype="multipart/form-data"

If you are using a mailto: link in the action attribute, your email body text will be less jumbled if you use

enctype="text/plain"

Chapter 4 :: Form Related Tags

Between the <form> and </form> tags are the tags that create the body of the form. These are <select> (for drop-down and menu boxes), <textarea> (for multi-line text areas), and <input> (for the rest).

Every form related tag can have a name attribute. Some require it, others don't.

The assigned name is sent with the information and helps the receiving program or function identify the information chunks it receives.
Some receiving programs and functions consider tag names to be case sensitive; some do not. If you don't know which applies to your form, assume tag names are case sensitive.

Some receiving programs and functions are okay with tag names containing spaces; some are not. If you don't know whether or not it is okay to use spaces in your form's tag names, use underscore characters instead of spaces. Underscore characters are almost always okay within tag names so long as the name doesn't start or end with an underscore.

Here is an example name attribute:

name="my_tag_name"

In most situations, it is highly desirable that no duplicate names exist within the form.

Chapter 5 :: The <input> Tag

Most of the information forms collect is specified by the <input> tag. Single-line text entry fields, checkboxes, radio selections, password entry fields, form buttons, file upload fields, and image buttons are all specified with the <input> tag.

The <input> tag can have several other attributes depending on the "type" attribute. It almost always has a name attribute. Every input tag requires a type attribute.

~ The "type" Attribute

It is probably safe to say that every form has an <input> tag of some kind and that the tag contains the "type" attribute. The type attribute tells the browser what type of form field or button to create. The type attribute looks something like this:

type="_____"

with the underscore representing one of numerous types of type attributes.

~ ~ The type="submit" Attribute

The submit button is the most common form element, created with the type="submit" attribute. It might or might not have a "value" attribute, depending on whether or not you want to specify the text of the button. Also, the submit button might or might not have a name, depending on what the information receiving program or function's requirements are.

In this example, it does not have a name but it does have a value:

```
<form method="POST" action="/cgi-bin/handler.cgi">
<input type="submit" value="Click Me!">
</form>
```

Forms may contain more than one submit button. In that case, it is usual for the attributes to have names so the processing program or function knows which of them was clicked.

Example:

```
<form method="POST" action="/cgi-bin/handler.cgi">
<input type="submit" name="1" value="First Visit">
<input type="submit" name="2" value="Second Visit">
<input type="submit" name="3" value="Third Visit">
</form>
```

~ ~ The type=”text” Attribute

The type=”text” attribute is a common form input field. It creates a place wherein a line of information can by typed.

If you want to put default text into the input field, use the “value” attribute. Example:

```
value=”http://”
```

Both a “size” and a “maxlength” attribute may be used. These specify the size of the text input area and the maximum number of characters that may be typed into the input area. You may use neither, one, or both of those tags.

Here is an example:

```
<form method="POST" action="/cgi-bin/handler.cgi">
<input type="text" name="url" size="17" maxlength="44">
<input type="submit">
</form>
```

~ ~ The type=”password” Attribute

The type=”password” attribute works the same as the type=”text” attribute except the characters typed into the input area are displayed as asterisks.

Here is an example:

```
<form method="POST" action="/cgi-bin/handler.cgi">
<input type="password" name="P" size="9" maxlength="20">
<input type="submit">
</form>
```

~ ~ The type=”checkbox” Attribute

The checkbox either contains information or it doesn’t, depending on whether or not it is checked. Checkbox tags almost always have a “value” attribute.

If you want any checkboxes pre-checked, include the CHECKED attribute. Example:

```
<form method="POST" action="/cgi-bin/handler.cgi">
<input type="checkbox" name="c_blue" value="yes">
<input type="checkbox" name="c_red" value="yes" CHECKED>
<input type="checkbox" name="c_pink" value="yes"
CHECKED>
<input type="checkbox" name="c_yellow" value="yes">
<input type="submit">
</form>
```

In the above example, the user will see four checkboxes with the middle two pre-checked.

Notice that each checkbox has a different name. This separates the information into different chunks for the receiving program or function to process. In some cases, the receiving program or function might require the checkboxes to have the same name, but do it way only if specified in the instructions.

~ ~ The type=”radio” Attribute

Only one radio button in a set contains information, the one that is checked. Radio tags almost always have a “value” attribute.

If you want to pre-check one radio button, use the CHECKED attribute:

```
<form method="POST" action="/cgi-bin/handler.cgi">
<input type="radio" name="color" value="blue">
<input type="radio" name="color" value="red" CHECKED>
<input type="radio" name="color" value="pink">
<input type="submit">
</form
```

In the above example, the user will see three radio buttons with the middle one pre-checked.

Each radio button in a set must have the same name. When more than one radio button has the same name, only one of them can be checked at a time.

~ ~ The type=”file” Attribute

If the form information receiving program or function can upload a file from the user’s computer to your internet server, the type=”file” attribute is used. (To use this attribute, the <form> tag should have the enctype attribute specified as “multipart/form-data”)

Optionally, you may restrict the type of file the user may upload with the “accept” attribute. The accept attribute specifies a MIME Content Type. There are many MIME types and to list them all would be space prohibitive. However, here are those you are most likely to need.

To restrict the uploads to images, use:

accept=”image/*”

To restrict to only GIF images, use:

accept=”image/gif”

To specify only JPEG, PNG, or TIFF image files, use “image/jpg,jpeg”, “image/png”, or “image/tiff”, respectively. To specify only GIF and JPEG, use “image/gif,jpg,jpeg”

To restrict to text files only, use:

accept=”text/*”

To specify HTML files or plain text files, use “text/html” or “text/plain”, respectively.

Optionally, you may specify the size of the input field and the maximum number of characters that may be entered with the “size” and “maxlength” attributes.

Here is an example file upload form input area:

```
<form method="POST"
      action="/cgi-bin/handler.cgi"
      enctype="multipart/form-data">
<input type="file" name="upfile" accept="image/*">
<input type="submit">
</form
```

~ ~ The type=”hidden” Attribute

Use the type=”hidden” attribute to specify information for the receiving program or function that the user may not change. This information is usually required by the program or function in order to process the other information and/or to determine what to sent back to the browser. The installation instructions of the program or function should specify what hidden fields are required, if any.

Here is an example:

```
<form method="POST" action="/cgi-bin/handler.cgi">
<input type="hidden"
       name="something"
       value="A line of content">
<input type="submit" value="Click For Next Page">
</form>
```

~ ~ The type=”reset” Attribute

This attribute creates a reset button. Clicking that button will reset the form to the state it was in when it was first loaded into the browser. It might or might not have a “value” attribute, depending on whether or not you want to

specify the text of the button. It rarely requires a name because the reset button’s value is rarely used by the receiving program or function.

In this example, it does not have a name but it does have a value:

```
<form method="POST" action="/cgi-bin/handler.cgi">
<input type="text" name="url" size="17" maxlength="44">
<input type="reset" value="Erase Everything!">
<input type="submit" value="Submit Now!">
</form>
```

~ ~ The type=”button” Attribute

This attribute creates a clickable button with text you specify in the “value” attribute. This type of button is typically used to send information to a JavaScript function with the “onClick” attribute. In this case, a submit button may be optional. Example:

```
<form method="POST" action="/cgi-bin/handler.cgi">
<input type="text" name="email" size="17" maxlength="44">
<input type="button"
      value="Give me a message"
      onClick="return alert('a message')">
<input type="submit">
</form>
```

~ ~ The type=”image” Attribute

An image may be used instead of a submit button. To do that, use the type=”image” attribute.

When using the type=”image” attribute, the “src” attribute must be specified. The src attribute is specified like you would specify the src attribute in an <img> tag to place an image on a web page. Optionally, you may include “align”, “border”, “height”, “width”, “hspace”, and/or “vspace” attributes, just like you would in an <img> tag.

When the user clicks the image, the form submits its information to the program or function specified in the <form> tag.

Here is an example:

```
<form method="POST" action="/cgi-bin/handler.cgi">
<input type="text" name="url" size="17" maxlength="44">
<input type="image"
      name="imageclick"
      src="myimage.gif"
      border="0">
</form>
```

Chapter 6 :: The <textarea> Tag

When asking for multi-line input from the form user, use the <textarea> tag.

The optional attributes “cols” and “rows” specify how many characters wide and how many lines deep to make the text area input field. If these are not specified, individual browsers will use their own default sizes.

The optional attribute “wrap” specifies what to do with lines that are longer than the width of the text area input field.

wrap=”off” specifies that the text is displayed exactly as typed. No lines will wrap.

wrap=”hard” specifies that any line longer than the width of the text area input field will wrap to the next line. When the information is sent to the processing program or function, the hard wraps will be sent with the information.

wrap=”soft” is the same as “hard” except the wrap applies only to the visual text area input field. The soft wraps are removed before the information is sent to the processing program or function.

Which wrap attribute you choose depends on the type of information the text area field will hold and your preference. If you’re asking for multi-line postal addresses, you’ll probably want “off” so the line-by-line formatting of the addresses are not compromised. For free-form text paragraphs, “soft” may be appropriate. If the line lengths must not exceed a certain number of characters (when pre-formatting for an outgoing email, for example), then “hard” is the logical specification.

If you do not specify a wrap attribute, wrap=”off” is assumed.

Anything between the <textarea> and </textarea> tags, including spaces or tabs, will be put into the text area field when the form is loaded. If you don’t want any default text, keep those two tags together.

```
<form method="POST" action="/cgi-bin/handler.cgi">
<textarea name="message" cols="20" rows="5"></textarea>
<input type="submit">
</form>
```

Chapter 7 :: The <select> Tag

The <select> tag is used to construct drop-down list boxes (sometimes called drop-down menus) and scrolling list boxes (sometimes called scrolling menus).

The <select> tag may contain a "size" attribute (which determines whether you get a drop-down or a scrolling list). It may also contain a MULTIPLE attribute (applicable only to scrolling lists). Both of these are optional.

If the size attribute is used, it's value determines how many lines are visible in the scrolling list box. If the size attribute is not used, you will have a drop-down list instead. Here is an example of a size attribute:

size="4"

If the MULTIPLE attribute is used, then the user can select multiple items in the scrolling list box. If it is not used, only one item can be selected.

The <select></select> tags contain a list of items. Each item on that list is specified with the <option></option> tags. (Examples of a drop-down list box and a scrolling list box are in the "The <option> Tag" section, below.)

~ The <option> Tag

Each <option></option> tag set contains one item for a drop-down list box or a scrolling list box.

The <option> tag may contain the SELECTED attribute to pre-select that item.

The "value" attribute may be optional. Most CGI programs require a value to work with the information to be processed. Some JavaScript functions do not.

Here are examples of a drop-down list box and a scrolling list box:

```
<form method="POST" action="/cgi-bin/handler.cgi">
<select name="myselect">
<option value="1">First</option>
<option value="2" SELECTED>Second</option>
<option value="3">Third</option>
<option value="4">Fourth</option>
<option value="5">Fifth</option>
</select>
<input type="submit">
</form>

<form method="POST" action="/cgi-bin/handler.cgi">
<select name="myselect" size="4" MULTIPLE>
<option value="1">First</option>
<option value="2" SELECTED>Second</option>
<option value="3">Third</option>
<option value="4" SELECTED>Fourth</option>
<option value="5">Fifth</option>
</select>
<input type="submit">
</form>
```

Chapter 8 :: A Complete Example

Forms collect information and/or present it. When the submit button is clicked, information is sent somewhere. Once the information is sent somewhere, the form has done its job.

The next two pages contain an example form using all of the tags mentioned in this tutorial. (There is no need to copy this form code- a link to download the example is provided after the example)

```
<form method="POST"
      action="/cgi-bin/formupload/handler.cgi"
      enctype="multipart/form-data">
<input type="hidden"
       name="something"
       value="A line of content">
<p>
What is your home page URL?<br>
<input type="text" name="url" size="17" maxlength="44">
</p>
<p>
What is your password?<br>
<input type="password" name="P" size="9" maxlength="20">
</p>
<p>
Select one or more colors that you like:<br>
<input type="checkbox" name="c_blue" value="yes">
<input type="checkbox" name="c_red" value="yes" CHECKED>
<input type="checkbox" name="c_pink" value="yes"
CHECKED>
<input type="checkbox" name="c_yellow" value="yes">
</p>
<p>
What is your favorite color?<br>
<input type="radio" name="color" value="blue">
<input type="radio" name="color" value="red" CHECKED>
<input type="radio" name="color" value="pink">
</p>
<p>
Upload any type of file:<br>
<input type="file" name="upfile">
</p>
<p>
<input type="button"
       value="Give me a message"
       onClick="return alert('a message')">
<!-- Note: onClick works only with JavaScript
          enabled browsers. -->
</p>
<p>
Tell me why you are here!<br>
<textarea name="message" cols="20" rows="5"></textarea>
</p>
<p>
What is your most <b><u>un</u></b>favorite color:<br>
```

```
<select name="s_unfav">
<option value="green">Green</option>
<option value="red">Red</option>
<option value="blue" SELECTED>Blue</option>
<option value="yellow">Yellow</option>
<option value="pink">Pink</option>
</select>
</p>
<p>
Select one or more colors you do not like:<br>
<select name="s_like" size="4" MULTIPLE>
<option value="green">Green</option>
<option value="red" SELECTED>Red</option>
<option value="blue" SELECTED>Blue</option>
<option value="yellow">Yellow</option>
<option value="pink">Pink</option>
</select>
</p>
<p>
<input type="image"
      name="imageclick"
      src="myimage.gif"
      border="0">
</p>
<p>
<input type="reset" value="Erase Everything!">
</p>
<p>
<input type="submit"
      name="1"
      value="This is my first visit">
</p>
<p>
<input type="submit"
      name="2"
      value="I've been here before">
</p>
</form>
```

A web page with the above form, along with a script written especially to handle the form's information, can be download at

http://willmaster.com/master/cgi-bin/arts/pl.pl?formtut

Decompress the downloaded ZIP file. (If you're uncertain about decompressing a ZIP file, the FAQ at http://MasterCGI.com/faq/ has information.)

Once decompressed, there will be four files: form.html, myimage.gif, handler.cgi, and readme.txt.

File form.html is the web page with the form. File myimage.gif is an image used by the form. File handler.cgi is the script to handle the information submitted by the form. And file readme.txt has preparation and installation instructions.

Installation is easy. Just a few simple steps.

When the form is used, the script will display a page with the form field information you supplied. It will also upload any file you specify into the same directory where the script is installed — the script does not check for the "accept" attribute.

Forms can be fun :)

With a little practice, you can be an expert!

William Bontrager

This **Peach-e-Book** is brought to you compliments of **WillMaster.com**.

The article first appeared as a two part tutorial in the **WillMaster Possibilities** weekly ezine.

You can get your free subscription to **WillMaster Possibilities** ezine by sending an email to
subscribe-possibilities@willmaster.com
or by visiting **the WillMaster Possibilities website.**

If you would like more information about CGI, you are invited to visit **MasterCGI.com**- the WebMaster's CGI briefing center.