By Glenn Fleishman

Form and function

The simplest—and really the only—method to get information from a visitor to a Web site is via an HTML form. Form tags appeared early in the HTML spec, and closely mirror or exactly duplicate familiar graphical user interface elements like radio buttons. Explaining how the input to forms is parsed by a server is conceptually difficult, but defining the pieces that go into forms is relatively easy.

The general form of a form begins with—well—with a FORM tag:

```
<FORM METHOD=“GET” ACTION=“URL”>
```

and ends with

```
</FORM>
```

The METHOD attribute defines how the information you’ve entered into the form gets from your browser to a program on the server that processes it. Three specifications exist: GET, PUT, and POST. GET turns the variables and values in your form into a long string of text that gets stuck at the end of a URL. This happens automatically, but it's ugly looking and can cause problems for many browsers and servers. PUT is just for uploading files to a server; clicking a "Submit" button in a form that uses PUT as its METHOD brings up a file selection dialog box on browsers that support it.

POST is the most commonly used METHOD, as it hides the information being sent "behind the scenes." The browser sends the variables and values directly to the server; the user doesn't see any of what's happening. The data are actually sent to a program at the server that can run separately from the server software itself, through the Common Gateway Interface (CGI), a standard method used by servers to send information to programs for processing.

ACTION specifies the location of the program that will process the form: it is always a URL (like *http://www.fosdick.com/cgi-bin/process-form*) that points to that program.

You can have multiple forms in an HTML page, one after another, but you can't nest them (that is, you have to write `<FORM...>form informa-`

`tion</FORM>`, then `<FORM...>form information</FORM>`, in sequence). Each form is treated as a unique set of data—only the input fields you have in a particular form are submitted when you click that form's "Submit" button.

Each of the kinds of input you request in a given form use the tag `<INPUT TYPE="text" NAME="anyname" VALUE="default value" SIZE="60,1">`

TYPE is necessary to define the kind of input you're asking for—radio button, checkbox, etc.—except in the case of pop-up menus or large text areas. NAME is the particular name you assign to the item of information you're asking for. In programming terms, NAME is just like a variable name. In non-programming terms, NAME is just an arbitrary label you assign to uniquely identify the value entered in the input field, or checked, or selected. VALUE is anything you want to use to prefill the field in question. And SIZE is the width and depth of the items. For all practical purposes, width is measured in fixed-font characters like Courier, and is the first of the pair; height is measured in rows.

Parts of the puzzle

Speaking broadly, there are only a few kinds of elements that are used in forms: text-entry fields, lists, and buttons (see Tables 1-1 and 1-2 on pages 12 and 13).

Text-entry fields. There are three kinds: simple text entry, "password" field entry, and text areas. A simple text-entry field is used most often to get user input, like the person's name or address; the user tabs from field to field. Password fields are used to enter information safe from prying eyes; these fields don't encrypt the contents, but they do hide the information while it's being entered.

Text areas are created by means of a paired tag. TEXTAREA doesn't use SIZE, but has separate attributes for defining, in characters and lines, width (COLS) and height (ROWS).

```
    <TEXTAREA ROWS=4 COLS=50 NAME="short essay"
WRAP="HARD">Enter essay here</TEXTAREA>
```

Text areas are essentially big fields that allow lots of text. The attribute

WRAP is a nice Netscape innovation that automatically wraps text (knocks the cursor down to the next line at the end of a line). In the example above, the words "Enter essay here," placed between the starting tag and the ending tag, prefill the text area; if the user doesn't write anything in this field, the words "Enter essay here" will be submitted when the user is done with the form.

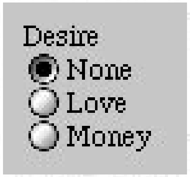
Lists. Checkboxes, radio buttons, pop-up menus, and scrolling lists are all options. These should be familiar from using any computer operating system.

Checkboxes allow "binary" choices: a given item is checked or not. Unlike FileMaker Pro's coding of checkboxes, which a lot of people are familiar with, each HTML checkbox is a separate variable—that is, a separately named field in which variable data can be entered. If you have ten options ("Send me the following brochures," for instance), you'll therefore have ten INPUT tags, each with a corresponding unique NAME.

Radio buttons always confuse people. While checkboxes may be checked or unchecked independent of each other, only one radio button in a category (defined by its NAME) can be selected at a time. So, if you have a question like

"Please select your age from the following: o 1–19 o 20–35 o 36–50 o 51+," obviously only one of those choices can be made, and that choice is exclusive of the three other options.

In pure interface design, you should always have one radio button in a set of options preselected (even if it's "o none"); however, most browsers don't enforce this requirement. To preselect a radio button (or, for that matter, a checkbox), you add the attribute CHECKED or CHECKED="ON" to the INPUT tag, like so:



```
<input type="radio" name="desire" value="none"
checked="on">
<input type="radio" name="desire" value="love" >
<input type="radio" name="desire" value="money" >
```

Pop-up menus and scrolling are a little trickier. Both of these are defined by using the SELECT tag, which indicates that the user is going to face a list:

```
<SELECT NAME="room color">
<OPTION NAME="green">A beautiful shade of green
```

```
<OPTION NAME="mauve">Really, don't choose mauve
<OPTION NAME="antique eggshell white" SELECTED>An
off-white, our most favored color
</SELECT>
```

Each item specified by OPTION appears in the pop-up menu. To make this same list into a scrolling list (where a few items are displayed and you can scroll through all options and then make multiple, noncontiguous selections), just add the MULTIPLE attribute. You further modify this with the SIZE attribute, which specifies how many items should appear in the list at once:

```
<SELECT NAME="room color" MULTIPLE SIZE="5">
```

Buttons. There are three kinds of buttons used in forms, and they're denoted by the same terms: SUBMIT, RESET, and IMAGE. The SUBMIT tag creates a button that, when clicked, sends the form, along with all the data the user has entered, off to the server to be chunked on. The RESET tag erases everything entered in the field—place and name this button wisely! The IMAGE attribute allows the use of an image in lieu of the "submit" button.

NAME defines the "Submit" button; VALUE is what's displayed inside the button (it's the same with "Reset").

You can have multiple "Submit" buttons and images with different names and values inside a single form. So a form could, at the bottom, say, "Click here to send via E-mail" and "Click here to send via courier pigeon." Your user has to choose one, but the data might be sent differently or with different consequences depending on which button is clicked on. This feature is used most frequently for behind-the-scenes activity: a programmer might "hide" values in a form that are invisible to the user except for the buttons for actions. The hidden values get passed to the server without the user having to do a lot of fumfering.

PageMill, SiteMill, and forms

All of the above is useful as background knowledge when you're planning the kind of input you need. But tedious hand coding is annoying.

Fortunately, PageMill and SiteMill really shine as form design programs. The 'Mills have a set of buttons at the top of each window that corresponds to each of the kinds of input types above. Instead of typing in ROWS and COLS for a text area, you can just drag the size of the area. With pop-up menus and lists, all you have to do is type in the various entries, one per line.

The forms are named and values are prefilled through the Attributes Inspector. First, create and size the input element; then click on it. The rightmost of three buttons in Attributes Inspector will light up. Click it, and you can enter name and value. Remember to hit "return" after entering each value!

The 'Mills insist that you preselect one radio button in each set. This is good form, but it's non-overrulable. If you choose not to select anything, you'll need to open up the form in a text editor after saving it in one of the 'Mills, and remove the CHECKED="ON" statement. (By default, this statement reappears each time you edit the page in PageMill or SiteMill.)

Processing forms

As I suggested at the beginning of this column, I'm going to give you a really inadequate answer to the question of how to process a form: you can't do it in

HTML, you need a CGI program to do it. Because the data in your form are always encoded before being transmitted to the server, the server program needs to decode them first, then process them. Fortunately, this isn't difficult, and lots of freeware decoders exist (some decoders usually ship with server software).

This is another one of those circumstances I define as "the system admin is your best friend" scenarios. Many companies and service providers have written generic processors for exactly this circumstance. Generally, all you need to use these processors is the HTML coding for your form (you'll probably have to include a few HIDDEN fields with specific values) and the location of the processing script (which is specified in the FORM tag's ACTION="URL").

This may appear unsatisfying. However, if you're lucky, your only role in this process will be to make the damned things, not to massage the information flow. A form designer does need to think about information coming out the other end—but doesn't need to suffer over how it goes from one to the other.

Forming a vision

Obviously, forms are useful and versatile animals—one of the foundations of HTML's utility. If you design your forms well, they will ensure focused responses from visitors. For tutorials and lots of reference material, see *http://www.yahoo.com/Computers_and_Internet/Internet/World_Wide_Web/Programming/Forms/*. ●

Go to Glossary ▸

Continued on page 12

Table 1-1

Input types and descriptions

Input type	Description	Special parameters
TEXT	Simple text field	
PASSWORD	Simple text field in which the input is replaced with bullets when you type, so as to hide the value (these are not encrypted, however)	
CHECKBOX	Checkbox	
VALUE	Can be set so that when the box is checked, what was coded into VALUE is returned for processing	
RADIO	Radio buttons	All radio buttons for each category must have the same name; the value is used to return the specific value of the selected radio button
SUBMIT	Creates a button that, when clicked, sends the contents of the form to the server	
RESET	Empties the form of all entered data!	Place cautiously. Easy to put in a place that allows the unwary to click and lose their entries
HIDDEN	Used mostly by programmers to pass values that users don't need to see, but that CGI scripts may need in order to process forms	
IMAGE	Allows an image to be used as a SUBMIT button	Has more sophisticated uses as an image map, but is rarely used in that way

Table 1-2

Related tags

Tag	Description	Special parameters
TEXTAREA	Defines a big entry space with rows and columns	ROWS and COLS are used to define space. Values are prefilled by entering between <TEXTAREA> and </TEXTAREA>
SELECT	Sets up a list of items for a pop-up menu or scrolling list	Values are noted by the <OPTION> tag and may be named individually

Go to Glossary ▸

Glossary entries

Variable. In programming, as in Web formatting, a variable is an arbitrarily named container that holds a value. Just as a beaker in a lab might be labelled "liquids" and hold at different times water, acid, or beer, so a variable names the container (like "username") and its contents are whatever value is assigned to it (like "Quentin Crisp"). In the context of forms, the variable name is that which is assigned by NAME="variable name" in the INPUT tag. *Back to article* ▸

Tag. Simply put, an HTML tag is the thing between the pointy brackets. So an anchor tag is <A HREF="URL">. Most tags come in pairs, with the opening tag having a set of attributes or modifiers, and the closing tag being simply a forward slash plus the tag's name. Tags that appear in pairs include headings, type styles (bold, italic, and so forth), and anchors. Tags that appear by themselves include the paragraph tag <P>, the horizontal rule tag <HR>, and the <INPUT> tag used in forms. *Back to article* ▸

Attribute. A tag has a name (like A for anchor) and then a variety of modifiers called attributes. For instance, with the IMG tag, used to insert an image into an HTML page, you have such attributes as SRC (the location of the image), WIDTH and HEIGHT (the pixel dimensions), and BORDER (defining the width of the hypertext border, if any). *Back to article* ▸