Creating the form

There are few differences between a straight HTML form and a JavaScript-enhanced form. The main one being that a JavaScript form relies on one or more event handlers, such as onClick or onSubmit. These invoke a JavaScript action when the user does something in the form, like clicking a button. The event handlers, which are placed with the rest of the attributes in the HTML form tags, are invisible to a browser that don't support JavaScript. Because of this trait, you can often use one form for both JavaScript and non-JavaScript browsers.

Typical form control objects -- also called "widgets" -- include the following:

- Text box for entering a line of text
- Push button for selecting an action
- Radio buttons for making one selection among a group of options
- Check boxes for selecting or deselecting a single, independent option

For use with JavaScript, you should always remember to provide a name for the form itself, and each control you use. The names allow you to reference the object in your JavaScript-enhanced page.

EX 1.

```
<FORM NAME="myform" ACTION="" METHOD="GET">

Enter something in the box: <BR>

<INPUT TYPE="text" NAME="inputbox" VALUE=""><P>

<INPUT TYPE="button" NAME="button" Value="Click" onClick="testResults(this.form)">

</FORM>
```

- *FORM NAME="myform"* defines and names the form. Elsewhere in the JavaScript you can reference this form by the name *myform*. The name you give your form is up to you, but it should comply with JavaScript's standard variable/function naming rules (no spaces, no weird characters except the underscore, etc.).
- *ACTION=""* defines how you want the browser to handle the form when it is submitted to a CGI program running on the server. As this example is not designed to submit anything, the URL for the CGI program is omitted.
- *METHOD="GET"* defines the method by which data is passed to the server when the form is submitted.
- *INPUT TYPE="text"* defines the text box object. This is standard HTML markup.
- *INPUT TYPE="button"* defines the button object. This is standard HTML markup except for the onClick handler.

- *onClick="testResults(this.form)"* is an event handler -- it handles an event, in this case clicking the button. When the button is clicked, JavaScript executes the expression within the quotes. The expression says to call the testResults function elsewhere on the page, and pass to it the current form object.

EX 2.

```
<HTML>

<HEAD>

<TITLE>Test Input</TITLE>

<SCRIPT LANGUAGE="JavaScript">

function testResults (form) {

    var TestVar = form.inputbox.value;

    alert ("You typed: " + TestVar);

}

</SCRIPT>

</HEAD>

<BODY>

<FORM NAME="myform" ACTION="" METHOD="GET">Enter something in the box:
<BR>

<INPUT TYPE="text" NAME="inputbox" VALUE=""><P>

<INPUT TYPE="button" NAME="button" Value="Click" onClick="testResults(this.form)">

</FORM>

</BODY>
```

```
</HTML>
```

Here's how the script works. JavaScript calls the testResults function when you click the button in the form. The testResults function is passed the form object using the syntax this.form (the this keyword references the button control; form is a property of the button control and represents the form object). I've given the form object the name *form* inside the testResult function, but you can any name you like.

The testResults function is simple -- it merely copies the contents of the text box to a variable named TestVar. Notice how the text box contents was referenced. I defined the form object I wanted to use (called *form*), the object within the form that I wanted (called *inputbox*), and the property of that object I wanted (the *value* property).

Setting a value in a form object

The value property of the inputbox, shown in the above example, is both readable and writable. That is, you can read whatever is typed into the box, and you can write data back into it. The process of setting the value in a form object is just the reverse of reading it. Here's a short example to demonstrate setting a value in a form text box. The process is similar to the previous example, except this time there are two buttons. Click the "Read" button and the script reads what you typed into the text box. Click the "Write" button and the script writes a particularly phrase into the text box.

EX3.

```
<HTML>

<HEAD>

<TITLE>Test Input </TITLE>

<SCRIPT LANGUAGE="JavaScript">

function readText (form) {

   TestVar =form.inputbox.value;

   alert ("You typed: " + TestVar);

}
```

```
function writeText (form) {

    form.inputbox.value = "Have a nice day!"

}

</SCRIPT>

</HEAD>

<BODY>

<FORM NAME="myform" ACTION="" METHOD="GET">

Enter something in the box: <BR>

<INPUT TYPE="text" NAME="inputbox" VALUE=""><P>

<INPUT TYPE="button" NAME="button1" Value="Read" onClick="readText(this.form)">

<INPUT TYPE="button" NAME="button2" Value="Write" onClick="writeText(this.form)">

</FORM>

</BODY>

</HTML>
```

- When you click the "Read" button, JavaScript calls the readText function, which reads and displays the value you entered into the text box.
- When you click the "Write" button, JavaScript calls the writeText function, which writes "Have a nice day!" in the text box.

Using Radio Buttons

Radio buttons are used to allow the user to select one, and only one, item from a group of options. Radio buttons are always used in multiples; there is no logical sense in having just one radio button on a form, because once you click on it, you can't unclick it. If you want a simple click/unclick choice use a check box instead (see below).

To define radio buttons for JavaScript, provide each object with the same name. JavaScript will create an array using the name you've provided; you then reference the buttons using the array indexes. The first button in the series is numbered 0, the second is numbered 1, and so forth. Note that the VALUE attribute is optional for JavaScript-only forms. You'll want to provide a value if you submit the form to a CGI program running on the server, however.

```
• <INPUT TYPE="radio" NAME="rad" VALUE="radio_button1" onClick=0>
• <INPUT TYPE="radio" NAME="rad" VALUE="radio_button2" onClick=0>
• <INPUT TYPE="radio" NAME="rad" VALUE="radio_button3" onClick=0>
• <INPUT TYPE="radio" NAME="rad" VALUE="radio_button4" onClick=0>
```

Following is an example of testing which button is selected. The for loop in the testButton function cycles through all of the buttons in the "rad" group. When it finds the button that's selected, it breaks out of the loop and displays the button number (remember: starting from 0).

```
<HTML>

<HEAD>

<TITLE>Radio Button Test</TITLE>

<SCRIPT LANGUAGE="JavaScript">

function testButton (form){

   for (Count = 0; Count < 3; Count++) {

      if (form.rad[Count].checked)

         break;

   }

   alert ("Button " + Count + " is selected");

}

</SCRIPT>
```

```
</BODY>

<FORM NAME="testform">

<INPUT TYPE="button" NAME="button" Value="Click"

   onClick="testButton(this.form)"> <BR>

<INPUT TYPE="radio" NAME="rad" Value="rad_button1" onClick=0><BR>

<INPUT TYPE="radio" NAME="rad" Value="rad_button2" onClick=0><BR>

<INPUT TYPE="radio" NAME="rad" Value="rad_button3" onClick=0><BR>

</FORM>

</HTML>
```

Setting a radio button selection with HTML market is simple. If you want the form to initially appear with a given radio button selected, add the CHECKED attribute to the HTML markup for that button:

```
<INPUT    TYPE="radio"    NAME="rad"    Value="rad_button1"    CHECKED
onClick=0>
```

You can also set the button selection programmatically with JavaScript, using the checked property. Specify the index of the radio button array you want to checked.

```
form.rad[0].checked = true;  // sets to first button in the rad group
```

Using Check Boxes

Check boxes are stand-alone elements; that is, they don't interact with neighboring elements like radio buttons do. Therefore they are a bit easier to use. Using JavaScript you can test if a check box is checked using the checked property, as shown here. Likewise, you can set the checked property to add or remove the checkmark from a check box. In this example an alert message tells you if the first check box is checked. The value is true if the box is checked; false if it is not.

```
<HTML>

<HEAD>

<TITLE>Checkbox Test</TITLE>

<SCRIPT LANGUAGE="JavaScript">

function testButton (form){

    alert (form.check1.checked);

}

</SCRIPT>

</BODY>

<FORM NAME="testform">

<INPUT TYPE="button" NAME="button" Value="Click"

    onClick="testButton(this.form)"><BR>

<INPUT TYPE="checkbox" NAME="check1" Value="Check1">Checkbox 1<BR>

<INPUT TYPE="checkbox" NAME="check2" Value="Check2">Checkbox 2<BR>

<INPUT TYPE="checkbox" NAME="check3" Value="Check3">Checkbox 3<BR>

</FORM>

</BODY>

</HTML>
```

As with the radio button object, add a CHECKED attribute to the HTML markup for that check box you wish to be initially check when the form is first loaded.

```
<INPUT TYPE="checkbox" NAME="check1" Value="0" CHECKED>Checkbox 1>
```

You can also set the button selection programmatically with JavaScript, using the checked property. Specify the name of the checkbox you want to check, as in

```
form.check1.checked = true;
```

Using Text Areas

Text areas are used for multiple-line text entry. The default size of the text box is 1 row by 20 characters. You can change the size using the COLS and ROWS attributes. Here's a typical example of a text area with a text box 40 characters wide by 7 rows:

```
<TEXTAREA NAME="myarea" COLS="40" ROWS="7">

</TEXTAREA>
```

EX.

```
<HTML>

<HEAD>

<TITLE>Text Area Test</TITLE>

<SCRIPT LANGUAGE="JavaScript">

function seeTextArea (form) {

    alert (form.myarea.value);

}

</SCRIPT>

</HEAD>

<BODY>

<FORM NAME="myform">

<INPUT TYPE="button" NAME="button3" Value="Test"
```

```
  onClick="seeTextArea(this.form)">

<TEXTAREA NAME="myarea" COLS="40" ROWS="5">

</TEXTAREA>

</FORM>

</BODY>

</HTML>
```

You can preload text into the text area in either of two ways. One method is to enclose text between the <TEXTAREA> and </TEXTAREA> tags. This method is useful if you wish to include hard returns, as these are retained in the text area box. Or, you can set it programmatically with JavaScript using the following syntax:

```
form.textarea.value = "Text goes here";
```

- *form* is the name of the form.
- *textarea* is the name of the textarea.
- *"Text goes here"* is the text you want to display

Using Selection Lists

List boxes let you pick the item you want out of a multiple-choice box. The listbox itself is created with the <SELECT> tag, and the items inside it are created by one or more <OPTION> tags. You can have any number of <OPTION> tags in a list. The list is terminated with a </SELECT> tag.

The list can appear with many items showing at once, or it can appear in a drop-down box -- normally you see one item at a time, but click to see more. The markup for the two styles is identical, except for the optional SIZE attribute. Leave off SIZE to make a drop-down box; use SIZE to make a list box of the size you wish.

Use the selectedIndex property to test which option item is selected in the list, as shown in the following example. The item is returned as an index value, with 0 being the first option, 1 being the second, and so forth (if no item is selected the value is -1).

```
<HTML>

<HEAD>

<TITLE>List Box Test</TITLE>

<SCRIPT LANGUAGE="JavaScript">

function testSelect(form) {

    alert (form.list.selectedIndex);

}

</SCRIPT>

</HEAD>

<BODY>

<FORM NAME="myform" ACTION="" METHOD="GET">

<INPUT TYPE="button" NAME="button" Value="Test" onClick="testSelect(this.form)">

<SELECT NAME="list" SIZE="3">

<OPTION>This is item 1

<OPTION>This is item 2

<OPTION>This is item 3

</SELECT>

</FORM>

</BODY>
</HTML>
```

If you want the text of the selected list item instead of the index, use this in the testSelect function:

```
function testSelect (form) {

   Item = form.list.selectedIndex;

   Result = form.list.options[Item].text;

   alert (Result);

}
```

Other events you can trigger within a form

I've used the onClick event handler in all of the examples because that's the one you are most likely to deal with in your forms. Yet JavaScript supports a number of other event handlers as well. Use these as the need arises, and the mood fits. In Navigator 2.x The event handlers used with form object are:

- *onFocus* -- an event is triggered with a form object gets input focus (the insertion point is clicked there).
- *onBlur* -- an event is triggered with a form object loses input focus (the insertion point is clicked out of there).
- *onChange* -- an event is triggered when a new item is selected in a list box. This event is also trigged with a text or text area box loses focus and the contents of the box has changed.
- *onSelect* -- an event is triggered when text in a text or text area box is selected.
- *onSubmit* -- an event is triggered when the form is submitted to the server (more about this important hander later in the column).

Submitting the form to the server

In the examples above I've limited the action of the form to within JavaScript only. Many forms are designed to send data back to a CGI program runnong on the server. This is referred to as submitting the form, and is accomplished using either of two JavaScript instructions: the onSubmit event handler or the submit method. In most instances, you use one or the other, *not both*!

- Place the onSubmit event hander in the <FORM> tag. This tells JavaScript what it should do when the user clicks the Submit button (this is a button defined as TYPE="submit").
- Place the submit instruction anywhere in your JavaScript. It can be activated by any action, such as clicking a form button that has been defined with the onClick event handler.

Using onSubmit

Here's an example of using the onSubmit event handler to send mail. The onSubmit event handler tells JavaScript what to do when the user clicks the Submit button: call the mailMe()

function, where additional mail fields are appended to a mailto: URL. Navigator automatically opens a new mail window with the fields filled in. Write the body of the message, and send the mail off to the recipient.

```
<HTML>

<HEAD>

<TITLE>onSubmit Test</TITLE>

<SCRIPT LANGUAGE="JavaScript">

function mailMe(form){

   Subject=document.testform.inputbox1.value;

   CC= document.testform.inputbox2.value;

   BCC= document.testform.inputbox3.value;

   location = "mailto:jwedit@javaworld.com?subject="+Subject+"&Bcc="+

      BCC+"&cc="+CC;

   return true;

}

</SCRIPT>

</HEAD>

<BODY>

<FORM NAME="testform" onSubmit="return mailMe(this.form)" >Subject of message:
<BR>

<INPUT TYPE="text" NAME="inputbox1" VALUE="This is such a great form!" SIZE=50>

<P>Send cc to: <BR>

<INPUT TYPE="text" NAME="inputbox2" VALUE="" SIZE=50><P>
```

```
Send blind cc to: <BR>

<INPUT TYPE="text" NAME="inputbox3" VALUE="" SIZE=50><P>

<INPUT TYPE="submit"><BR>

</FORM>

</BODY>

</HTML>
```

Using submit

In the next example the submit method is used instead. The script is little changed, except that the onSubmit handler is removed, and an onClick hander for a renamed form button is added in its place. The submit() method replaces the return true statement in the previous example. (Personally, I prefer the submit method because it provides a little more flexibility. But either one will work.)

```
<HTML>

<HEAD>

<TITLE>test</TITLE>

<SCRIPT LANGUAGE="JavaScript">

function mailMe(form){

  Subject=document.testform.inputbox1.value

  CC= document.testform.inputbox2.value

  BCC= document.testform.inputbox3.value

  location = "/javaworld/cgi-bin/jw-mailto.cgi?jwedit@javaworld.com?subject="+Subject+"&Bcc="+

    BCC+"&cc="+CC
    document.testform.submit();

}

</SCRIPT>

</HEAD>

<BODY>

<FORM NAME="testform">

Subject of message: <BR>

<INPUT TYPE="text" NAME="inputbox1" VALUE="This is such a great form!"
SIZE=50><P>

Send cc to: <BR><INPUT TYPE="text" NAME="inputbox2" VALUE="" SIZE=50><P>

Send blind cc to: <BR><INPUT TYPE="text" NAME="inputbox3" VALUE="" SIZE=50><P>

<INPUT TYPE="button" VALUE="Send Mail" onClick="mailMe()"><BR>

</FORM>

</BODY>

</HTML>
```

Validating form data using JavaScript

The World Wide Web "grew up" when they added the ability to display forms. In the days before forms, the Web was only mildly interactive, with just hypertext links to take readers from location to location. Forms allow users to truly interact with the Web. For example, readers can specify search queries using a simple one-line text box.

The typical form as used on a Web page consists of two parts: the form itself, which is rendered in the browser, and a CGI script or program located on the server. This script processes the user's input. While it's not exactly rocket science, a stumbling block in creating great Web forms is writing the CGI program. In most cases these programs are written in Perl or C, and can be a bother to implement and debug. A primary job of the CGI program is to validate that the reader has provided correct data, and this can requires pages of code.

JavaScript changes that. With JavaScript you can check the data provided by the reader before it's ever sent to the CGI program. In this way the CGI program can be kept to a bare minimum. And, because the data is only sent after it has been validated, the server need not be bothered until the form entry is known to be good. This saves valuable server resources.

Most form validation chores revolve around basic data checking: does the user remember to fill in an box? Is it have the right length? Does it contain valid characters? With most forms you can readily answer these questions with a small handful of validation routines.

A typical validation routine is determining if an input box contains only numeric digits, shown below. If the entry contains non-numeric characters, you can ask the user to enter the correct data. A ready-made routine for this is the isNumberString function, which returns the value 1 if the string contains only numbers, and 0 if it contains any non-numeric characters. To use it, provide the data string as the parameter. The value returned by the function tells you if the data is valid.

```
<HTML>

<HEAD>

<TITLE>Test Input Validation</TITLE>

<SCRIPT LANGUAGE="JavaScript">

function testResults (form) {

   TestVar = isNumberString (form.inputbox.value)

   if (TestVar == 1)

       alert ("Congratulations! You entered only numbers");

   else

       alert ("Boo! You entered a string with non-numbers characters");

}

function isNumberString (InString)  {

   if(InString.length==0) return (false);

   var RefString="1234567890";
```

```
    for (Count=0; Count < InString.length; Count++)  {

        TempChar= InString.substring (Count, Count+1);

        if (RefString.indexOf (TempChar, 0)==-1)

            return (false);

    }

    return (true);

}

</SCRIPT>

</HEAD>

<BODY>

<FORM NAME="myform">

Enter a string with numbers only:

<INPUT TYPE="text" NAME="inputbox" VALUE="">

<INPUT TYPE="button" NAME="button" Value="Click" onClick="testResults(this.form)" >

</FORM>

</BODY>

</HTML>
```