Forms require at least two files:
- the web page holding the form
- the web page that processes the form data

Form editing and processing is done with script (either JavaScript or VBScript); functions may reside in additional script files.

Contents:

Actual Form Web Page

The form page contains several components:
- text (as paragraphs, lists, etc.) and images
- the actual form elements (controls)
- script that positions cursor in a control when form opens
- scripts that edit entered data
- submission

There are several HTML tags used exclusively for forms: The paired FORM tag defines the entire form. Nested within it are tags for individual controls: INPUT, BUTTON, SELECT, OPTION, OPTGROUP, TEXTAREA, FIELDSET, LEGEND. Some controls may be labeled with the LABEL tag.

Sample:
```
<FORM . . .>
  <INPUT . . .>
  <INPUT . . .>
</FORM>
```

Form Controls

There are several types of controls:
- text box
- command button
- check box, radio button
- file select (allows user to select a file to be submitted with the form; presented with Browse button which opens a “Choose file” window)
- menu

There are four kinds of command buttons. Three have predefined actions: *Submit button* activates the form submission. *Image button* is a graphical submit button that is decorated with a custom image. *Reset button* resets the controls to their original values. *Push button* has an event procedure defined in script.

Several HTML tags define controls: INPUT, BUTTON, SELECT, OPTION, OPTGROUP, TEXTAREA, FIELDSET, LEGEND. Tags that are paired have content, text and/or image.

LABEL

Some form controls automatically have labels associated with them (e.g., push buttons), while most do not. The paired LABEL tag is used to specify labels for controls that do not implicitly have them. The tag has attribute FOR which identifies the control to be labeled by its id. It also has general attributes ID, CLASS, STYLE, and various event attributes (e.g., ONCLICK). The label is presented to the left or right of the control depending if its tag occurs before or after the control's tag.

Sample:
```
<FORM . . .>
  <LABEL FOR=me>Label text</LABEL>
  <INPUT TYPE=text SIZE=20 ID=me>
</FORM>
```

Labels can also be done with HTML content:
```
Last name: <INPUT NAME=lastname TYPE=text SIZE=15 TABINDEX=1>
<BR>
First name: <INPUT NAME=firstname TYPE=text SIZE=20 TABINDEX=2>
```

INPUT

Most form controls are defined by the unpaired **INPUT tag**. It has form-specific attributes TYPE, NAME, VALUE, SIZE, MAXLENGTH, CHECKED, SRC, and TABINDEX. It also has general attributes ID, CLASS, STYLE, and various event attributes (e.g., ONCHANGE).

The TYPE attribute defines the control type. Values are: text (single-line), password, checkbox, radio, button, hidden, file, submit, reset, image. Hidden controls are not rendered, but their values are submitted with the form.

The NAME attribute assigns the control name.

The VALUE attribute specifies the control's initial value. It is optional except for radio buttons and check boxes.

The SIZE attribute specifies the initial width of the control. The unit of measure is pixels except for text, password, and file select whose unit of measure is character.

The MAXLENGTH attribute specifies the maximum number of characters that the user may enter. Applies only to text and password controls. When the value is greater than SIZE, the browser provides a scrolling mechanism.

The CHECKED attribute, which applies only to radio buttons and check boxes, specifies if the control is selected.

The SRC attribute applies only to the image button. It specifies the location of the image file.

The TABINDEX attribute specifies the position of the subject control in the tab order. Avoid leading zeroes.

Radio buttons present a set of mutually exclusive choices. When one is one, all others are off. All radio buttons in the same group-set have the same NAME but different VALUE. Only one radio button can have CHECKED = True.

Example:
```
<INPUT  TYPE=radio  NAME=biztype  VALUE=”mfg”>Manufacturer<BR>
<INPUT  TYPE=radio  NAME=biztype  VALUE=”con”>Consultant<BR>
<INPUT  TYPE=radio  NAME=biztype  VALUE=”dis”>Distributor<BR>
```

BUTTON

The paired **BUTTON tag** defines a command button. It has attributes: TYPE, NAME, VALUE, and TABINDEX. It also has general attributes ID, CLASS, STYLE, and various event attributes (e.g., ONCLICK). Tag content can be text and/or image. Buttons created with this tag function like those created with the INPUT tag, but they offer richer rendering possibilities.

The TYPE attribute has values submit, reset, and button. The NAME attribute assigns the control name. The VALUE attribute specifies the control’s initial value; however it is not displayed with IE5. The width of the button is enough to present the contents (the text and/or image within the paired tags) with margins.

Sample:
```
<BUTTON TYPE=button>button text goes here</BUTTON>
<BUTTON TYPE=button>this button text goes here<IMG SRC=”logol.gif”>
   </BUTTON>
```

SELECT, OPTION, and OPTGROUP

The paired **SELECT tag** creates a menu or list. Each choice is specified by a nested OPTION tag of which there must be at least one. Choices can be grouped by the paired OPTGROUP tag.

The SELECT tag has attributes NAME, SIZE, MULTIPLE, and TABINDEX. It also has general attributes ID, CLASS, STYLE, and various event attributes (e.g., ONCLICK). The SIZE attribute specifies the number of rows to be visible. The MULTIPLE attribute specifies one or more selections are allowed.

In IE5, the menu choices are listed sequentially in a list box sized by the SIZE attribute and whose bottom is on the baseline; this alignment can be changed with CSS vertical-align. When the number of options > size, a vertical scroll bar appears.

The paired OPTION tag specifies a selectable choice for a SELECT tag. It has attributes SELECTED, VALUE, and LABEL. The SELECTED attribute specifies the choice is pre-selected. The VALUE attribute specifies the initial value of the control; if it is not used, the initial value is set to the contents of the tag. The LABEL attribute overrides the browser's use of the tag's content as a label; however this doesn't work in IE5.

Sample:
```
<SELECT SIZE=1 MULTIPLE>
  <OPTION>first choice</OPTION>
  <OPTION>second choice</OPTION>
</SELECT>
```

The paired OPTGROUP tag defines a group of choices. It has attribute LABEL which defines the label for the group of choices. Subject choices are grouped within the OPTGROUP tags. OPTGROUPS may not be nested. The group's label is presented differently (bold and italic) than choices, which are indented. Some of this formatting can be controlled with CSS.

Sample:
```
<SELECT SIZE=8>
  <OPTGRPOUP LABEL=first>
    <OPTION>first choice</OPTION>
    <OPTION>second choice</OPTION>
  </OPTGROUP>
</SELECT>
```

TEXTAREA

The paired **TEXTAREA tag** creates a multi-line text box. It has attributes NAME, ROWS, COLS, and TABINDEX. It also has general attributes ID, CLASS, STYLE, and various event attributes (e.g., ONCHANGE). IE5 always presents a vertical scroll bar (and no horizontal one).

Attribute NAME assigns the control's name.

Attribute ROWS specifies the number of visible text lines. When user enters more text than fits in the number of rows, a scroll bar appears.

Attribute COLS specifies the width of the control in average character widths. When user enters more text than fits in the number of rows, a scroll bar appears.

Sample:
```
<TEXTAREA ROWS=2 COLS=50>
First line of text.
Second line of text.
</TEXTAREA>
```

FIELDSET and LEGEND

The paired **FIELDSET tag** creates a box that surrounds embedded controls and labels. This allows authors to visually group related controls. The paired LEGEND tag creates a caption for the group box that, in IE5, appears on top of the top border.

The FIELDSET tag has only event attributes (e.g., OnMouseOver). The width of the fieldset defaults to the window width; this can be changed with CSS.

The LEGEND tag has no unique attributes. It has general attributes ID, CLASS, STYLE, and various event attributes (e.g., ONCLICK). The position of the legend relative to the fieldset can be specified with a deprecated attribute (ALIGN) or with CSS text-align and vertical-align; however, none of these have any effect in IE5. The default position is top and left.

Sample:
```
<FIELDSET>
<LEGEND>contents of legend</LEGEND>
<INPUT TYPE=text SIZE=30>
</FIELDSET>
```

Scripts

Client-side scripts can:
- size and position web page when it opens
- position cursor when form opens
- edit entered data
- manipulate one control based on the value of a second control
- determine which, if any, value was selected for a SELECT control
- set style rules
- disable/enable controls
- present help

Event procedures are one mechanism for implementing client-side scripts. There are several ways in which an event procedure can be executed:

- Within an HTML tag:
  ```
  <INPUT . . . ONCHANGE="function_name"
  <INPUT . . . ONCHANGE="statement(s)"
  ```

- Separate from HTML tag:
  ```
  <INPUT NAME=field2 SIZE=50>
  <SCRIPT TYPE=text/vbscript>
  Sub field2_changed()
  If field2.value = "abc" Then
     button3.enabled = True
  Else
     button3.enabled = False
  End If
  End Sub
  </SCRIPT>
  ```

A page's OnOpen event can position the cursor in the first form control.
```
<head>
<script type="text/javascript">
// puts focus on first element in first form
function setfocus(a,b)
```

```
{  document.forms[a].elements[b].focus()  }
</script>
</head>
<body  onLoad=”setfocus(0,0)”>
<form>
<input type="text" name="field" size="30">
<input type="text" name="userid" size="10">
<input type="button" value="Get Focus">
</form>
</body>
```

A control’s double-click (OnDblClick) event can open a help file.

A control’s OnChange event can edit the value.

A control’s event can affect other controls:

- Give focus to a second control by using the *focus* method (see positioning cursor when form opens).
- Disable a second control by setting its *disabled* attribute to True. A disabled control does not get focus, is skipped by tabbing navigation, and cannot be successful. Most controls can have the disabled attribute: button, input, optgroup, option, select, and textarea.
- Protect a second control by setting its *readonly* attribute to True. A readonly control can have focus, is included in tabbing navigation, can be successful, but cannot be changed by the user. Two controls have the readonly attribute: input and textarea.

Determine which, if any, value was selected for a SELECT control:

```
if (document.soq.state.selectedIndex == -1)       // none selected
```

In JavaScript, the SELECT element can have one of two types: “select-one” indicates only a single selection is allowed, and “select-multiple” indicates one or more selections are allowed. If multi-select, property selectedIndex is the index of the first selected entry.

Style rules can be set. Elements subject to this should have an id established in their HTML tag. If elements are to be handled as group, they can be grouped by the DIV tag. Script code then sets the element’s style:

```
<DIV id=”x”> . . . </DIV>
. . .
var eid = document.getElementById(“x”);
eid.style.display = “none”;
```

User input can be blocked by disabling a control (applicable to BUTTON, INPUT, OPTGROUP, OPTION, SELECT, and TEXTAREA) or making the control read-only (applicable to INPUT and TEXTAREA). Disabled controls cannot receive focus and cannot be successful. Read-only controls can get the focus and can be successful. These properties can be modified dynamically in script, but there are some limitations. You can disable a button:

```
document.soq.cont.disabled = true;
```

The form’s OnSubmit event can initiate editing.

Server-side scripts process the form data after it is submitted. They are discussed in the “Form Processing Web Page” section.

Submission

Form data is sent to a separate web page by submission. This is specified by the FORM tag.

FORM

The paired FORM tag acts as a container for form controls. It has attributes ACTION, METHOD, ENCTYPE, ACCEPT-CHARSET, ACCEPT, and NAME. It also has general attributes ID, CLASS, STYLE, and various event attributes (e.g., ONSUBMIT, ONRESET). The ONSUBMIT and ONRESET can return *False* and cancel the submission or reset.

The ACTION attribute specifies the URL of the processing web page.

The METHOD attribute has two values: get and post. "Get" is the default. "Get" causes the form data to be appended to the URL specified by the ACTION attribute and separated from it by a question mark ("?"). "Post" causes the form data to be included in the header of the form page and sent to the URL. If the form processing modifies a database, the "post" method should be used. The "get" method restricts data values to ASCII characters; only the "post" method covers the entire character set.

The ENCTYPE attribute is used only when the METHOD = post. It specifies the content type used to submit the form to the server. The value "application/x-www-form-urlencoded" is the default value. The value "multipart/form-data" should be used when an INPUT tag has TYPE=file.

The ACCEPT-CHARSET attribute specifies the list of character encodings for data sent to the server. The value is a space-and/or comma-delimited list of charset values. The default value is "unknown" which is adequate.

The ACCEPT attribute specifies a comma-separated list of content types that a server processing this form will handle correctly. This is optional.

The NAME attribute assigns the element a name that can be referred to by scripts and/or style sheets. However, the ID attribute should be used for this purpose.

Form Data

Only successful controls are valid for submission.

- Controls with no value are not successful. (N/A for text and select in IE6.)
- Disabled controls are not successful, but hidden controls may be successful.
- If a form contains more than one submit button, only the activated one is successful.
- Reset buttons are not successful.
- All selected checkboxes may be successful.
- Only selected radio buttons may be successful.
- Only selected list box choices may be successful.
- The current value of a file select control is a list of one or more file names. Upon submission, the contents of each file are submitted with the rest of the form data.

A form data set is a sequence of control-name:value pairs constructed from successful controls. Example:
`firstname=aaa&lastname=bbb&userid=ccc&sub=ddd&topic=eee&topic=fff.`

Sample Form

The **onSubmit** event handler invokes the editing logic. The form is submitted to the file named in **action** only if the event handler returns *true*.

```
<body>
<FORM action="sub.html" method="get" onSubmit=”return CheckForm(this)”>
<ol>
<li>Identify yourself.<BR>
First name: <input type="text" name="firstname"><BR>
Last name: <input type="text" name="lastname"><BR>
E-mail userid: <input type="text" name="userid" size=4> @ PGE.COM<BR>
<BR>
<li>Choose one desired action.<BR>
    <input type="radio" name="sub" value="subscribe"> Subscribe<BR>
    <input type="radio" name="sub" value="cancel"> Cancel
    subscription<BR>
    <input type="radio" name="sub" value="change"> Change
    subscription<BR>
<BR>
<li>Select topic(s) for which you want notification.<BR>
    <input type="checkbox" name="topic" value="intro"> 1.
    Introduction<BR>
    <input type="checkbox" name="topic" value="common"> 3. Common<BR>
    <input type="checkbox" name="topic" value="mtrmgmt"> 5. Meter
    Management<BR>
    <input type="checkbox" name="topic" value="mtrread"> 6. Meter
    Reading<BR>
    <input type="checkbox" name="topic" value="field"> 7. Field
    Work<BR>
<BR>
<li>Submit to Manual editor.<BR>
     <input type="submit" value="Submit" style="background: teal"><BR>

</ol>
</FORM>
```

The Submit control initiates a link to URL:
```
sub.html?firstname=aaa&lastname=bbb&userid=ccc&sub=ddd&topic=eee&topic=
    fff
```
where xxx is the value entered/selected/checked in the various input fields. Note: there are as many “topic” pairs as there are checked controls. Only successful controls are sent as name-value pairs. IE has a max URL length = 2,048 characters; this applies to both POST and GET request URLs.

Form Editing Logic

This code may be held in the same page as the form, however it is preferred to put it in a separate script file.

```
<head>
<script type="text/javascript">
function RequiredPresent(field)
{
    if (field.value == "") { return false }
    else                   { return true }
}

function CheckForm(input)
{
    var ErrorCount = 0
    var msg = " is required, please enter before submitting
   subscription request."
    var field = input.elements["firstname"]
    if (RequiredPresent(field) == false)
       {
        ++ErrorCount
        field.focus()
        alert("First name" + msg)
       }
    field = input.elements["lastname"]
    if (!RequiredPresent(field))
       {
        ++ErrorCount
        field.focus()
        alert("Last name" + msg)
       }
    field = input.elements["userid"]
    if (!RequiredPresent(field))
       {
        ++ErrorCount
        field.focus()
        alert("E-mail userid" + msg)
       }
    var validid = /\w{4}/
    if (field.value != "")
       {
        if (validid.test(field.value) != true)
           {
            ++ErrorCount
            field.focus()
            alert("E-mail userid must be four alphanumeric
   characters.")
           }
       }

    var CountChapters = input.topic.length
    for (var n = 0; n < CountChapters; n++)
       {
        if (input.topic[n].checked)   { break }
       }

    if (n == CountChapters)
       {
        ++ErrorCount
        alert("At least one chapter must be selected.")
       }

    if (ErrorCount > 0) { return false }
    else                { return true }
}
</script>
```

Form Processing Web Page

This page must parse the URL string to retrieve the form data. In the following example this is done with a JavaScript file urlparser.js which I downloaded from wevbdeveloper.earthweb.com.

After parsing the URL string into two arrays—one for the names, the second for the values—the program uses the data to populate explicitly-named variables. These variables are then used for specific purposes, in this case to create a e-mail.

```
<head>
<script type="text/javascript" src="../urlparser.js">
</script>

<script type="text/javascript">
<!--
// document.URL is mentioned in reference docs, but it excludes the ?
    portion
parseCallingURL(window.location.href)     //function is in urlparser.js

var idx = getMaxVars()                    //function is in urlparser.js
var na = getNameArray()                   //function is in urlparser.js
var va = getValueArray()                  //function is in urlparser.js
var FirstName = null
var LastName = null
var Userid = null
var Action = null
var TopicList = ""
var n
for (n=0; n<idx; n=n+1)
{ switch (na[n])
  { case "firstname" : FirstName = va[n]; break;
    case "lastname" :  LastName  = va[n]; break;
    case "userid" :    Userid    = va[n]; break;
    case "sub" :       Action    = va[n]; break;
    case "topic" :     TopicList = TopicList + va[n] + ", "; break;
    default :          document.write("<p>Sorry!  Program error.
    Notify Manual editor.</p>")
   }
}

var editor = "xxxx@pge.com"
var intro = "<p>Please confirm the following request to be
    automatically notified of \
changes in the Soft Tables Data Maintenance Manual. \
Correct any information as necessary and forward this e-mail to the
    editor ("
+ editor + ").</p>"
var you = "<p>First name = " + FirstName +
         "<br>Last name = " + LastName +
         "<br>Userid = " + Userid + "@pge.com</p>"
var act = "<p>Action requested = " + Action + "</p>"
var topics = "<p>Topics = " + TopicList

document.write (intro + you + act + topics)

//-->
</script>
</head>
```