Content Designing using HTML (Frames and Forms)

1. Frameset

An HTML document that describes frame layout (called a frame-set document) has a different makeup than an HTML document without frames. A standard document has one HEAD section and one BODY. A frame-set document has a HEAD, and a FRAMESET in place of the BODY.

The FRAMESET section of a document specifies the layout of views in the main user agent window. In addition, the FRAMESET section can contain a NOFRAMES element to provide alternate content for user agents that do not support frames or are configured not to display frames.

Elements that might normally be placed in the BODY element must not appear before the first FRAMESET element or the FRAMESET will be ignored. The attributes of this element are:

(i) rows = multi-length-list
   This attribute specifies the layout of horizontal frames. It is a comma-separated list of pixels, percentages, and relative lengths. The default value is 100%, meaning one row.

(ii) cols = multi-length-list
    This attribute specifies the layout of vertical frames. It is a comma-separated list of pixels, percentages, and relative lengths. The default value is 100%, meaning one column.

The browser might approximate some values to make the total height of the rows equal to the height of the window or the total width of the columns equal to the width of the window.

You can use any of the following measurements to specify values for ROWS and COLS:

(i) Size of a frame in pixels: If you specify the size of one or more frames in pixels, you should specify the size of other frames in the FRAMESET as a relative value. Otherwise, the browser can override the pixel values you specify to ensure that the size of the frames is proportional to the size of the browser window.

(ii) Percentage size of each frame: If you specify the frame sizes as percentages, the total of all values should be 100%. If the total is greater than 100%, all frames are scaled down. If the total is less than 100% and relative sized frames exist in the FRAMESET, the extra size is distributed among them; otherwise, all the frames are scaled up in size.

(iii) Relative size of each frame: You can specify the size of the first frame as a fixed number of pixels, and the size of the other frames become relative to the available remaining space.
Setting the rows attribute defines the number of horizontal subspaces in a frameset. Setting the cols attribute defines the number of vertical subspaces. Both attributes may be set simultaneously to create a grid. If the rows attribute is not set, each column extends the entire length of the page. If the cols attribute is not set, each row extends the entire width of the page. If neither attribute is set, the frame takes up exactly the size of the page.

Frames are created left-to-right for columns and top-to-bottom for rows. When both attributes are specified, views are created left-to-right in the top row, left-to-right in the second row, etc.

The first example divides the screen vertically in two (i.e., creates a top half and a bottom half).

```
<FRAMESET rows="50%, 50%">
...the rest of the definition...
</FRAMESET>
```

The next example creates three columns: the second has a fixed width of 250 pixels (useful, for example, to hold an image with a known size). The first receives 25% of the remaining space and the third 75% of the remaining space.

```
<FRAMESET cols="1*,250,3*">
...the rest of the definition...
</FRAMESET>
```

The next example creates a 2x3 grid of subspaces.

```
<FRAMESET rows="30%,70%" cols="33%,34%,33%">
...the rest of the definition...
</FRAMESET>
```

For the next example, suppose the browser window is currently 1000 pixels high. The first view is allotted 30% of the total height (300 pixels). The second view is specified to be exactly 400 pixels high. This leaves 300 pixels to be divided between the other two frames. The fourth frame's height is specified as "2*", so it is twice as high as the third frame, whose height is only "*" (equivalent to 1*). Therefore the third frame will be 100 pixels high and the fourth will be 200 pixels high.

```
<FRAMESET rows="30%,400,*,2*">
...the rest of the definition...
</FRAMESET>
```

The user agent should adjust absolute lengths that do not sum to 100% of the real available space. When underspecified, remaining space should be allotted proportionally to each view. When overspecified, each view should be reduced according to its specified proportion of the total space.
2. Frame

The FRAME element defines the contents and appearance of a single frame.

(i) src

The src attribute specifies the initial document the frame will contain. The following example shows a page with frames:

```html
<html>
<head>
<title>First FrameSet Page</title>
</head>
<br>
<frameset cols=50%,50%>
   <frameset rows=65%,35%>
      <frame src=html.htm>
      <frame src=java.htm>
   </frameset>
   <frameset rows=50%,50%>
      <frame src=vbscript.htm>
      <frame src=javascript.htm>
   </frameset>
</frameset>
</html>
```

This will generate a screen as follows:

![Figure 1](image)

(ii) marginwidth = n and marginheight= n

marginheight and marginwidth specify a margin in pixels that separates the frame content. Margins must be one pixel or greater, and they must allow enough space to display the frame content properly. If you do not specify MARGINHEIGHT and MARGINWIDTH, the browser determines the appropriate margin size.
a. scrolling = auto | yes | no

This attribute specifies scroll information for the frame window. Possible values

- auto: This value tells the user agent to provide scrolling devices for the
  frame window when necessary. This is the default value.
- yes: This value tells the user agent to always provide scrolling devices for
  the frame window.
- no: This value tells the user agent not to provide scrolling devices for the
  frame window.

(iii) noresize

When present, this boolean attribute tells the user agent that the frame window
must not be resizeable.

The following example creates a set of two frames.

```html
<FRAMESET COLS="30%,70%">
  <FRAME SRC="toc.html" NAME ="listFrame">
  <FRAME SRC="topic.html" NAME ="contentFrame">
</FRAMESET>
```

The two frames appear as columns because COLS is specified within the FRAMESET
stag. The left frame uses 30% of the available space, and the right frame uses the
remaining 70% of the space. By default, the frames in this example have scroll bars
and are resizeable, because no values are specified for the SCROLLING and
NORESIZE attributes.

(iv) target = frame-target [CI]

This attribute specifies the name of a frame where a document is to be opened.
By assigning a name to a frame via the name attribute, authors can refer to it as the
"target" of links defined by other elements. The target attribute may be set for elements
that create links (A, LINK), image maps (AREA), and forms (FORM).
This example illustrates how targets allow the dynamic modification of a frame's
contents. First we define a frameset in the document, shown here:

```html
<HTML>
<HEAD><TITLE></TITLE></HEAD>
<FRAMESET cols=25%,75%>
  <FRAME src="lang.htm">
  <FRAME name = "detail" src="finder2.jpg">
</FRAMESET>
</HTML>
```

Then, in lang.html, we link to the frame named "detail".

```
<HTML>
```
Activating either link opens a new document in the frame named "detail".

**Note.** A frameset definition never changes, but the contents of one of its frames can. Once the initial contents of a frame change, the frameset definition no longer reflects the current state of its frames.

(5) **NOFRAMES**

The NOFRAMES element specifies content that should be displayed only when frames are not being displayed. User agents that support frames must only display the contents of a NOFRAMES declaration when configured not to display frames. User agents that do not support frames must display the contents of NOFRAMES in any case.

NOFRAMES can be used in the FRAMESET section of a frameset document.

For example:

```html
<HTML>
<HEAD>
</HEAD>
```

Training Programme under CAFT “Online Content Creation and Management in an eLearning Environment”
3. Forms

You can design web pages that look like forms. Using forms, your server can interact with users to gather information. Users can browse through the forms, making selections and entering data as they move within the form. In a form, users can select from several choices, using a variety of selection methods, such as buttons, fill in the blanks, and selection lists. The way the users select any information depends on the form elements you use when you create the form. This section describes how to implement the following elements in a form:

3.1 Creating Forms

3.1.1. <FORM>

The <FORM> tag creates an HTML form, which lets users input text and make choices from elements such as checkboxes, radio buttons, and selection lists. A user fills out the form, and then submits the form by clicking a button. The way your browser or the server handles your form depends on the way you specify attributes to your form.

- action = uri

This attribute specifies a form-processing agent. For example, the value might be an HTTP URI (to submit the form to a program) or a mailto URI (to email the form).

- method = get | post

This attribute specifies which HTTP method will be used to submit the form data set. Possible (case-insensitive) values are "get" (the default) and "post".

- get: With the HTTP "get" method, the form data set is appended to the URI specified by the action attribute and this new URI is sent to the processing agent.

- post: With the HTTP "post" method, the form data set is included in the body of the form and sent to the processing agent.
The "get" method should be used when the form is idempotent (i.e., causes no side effects). Many database searches have no visible side effects and make ideal applications for the "get" method.

If the service associated with the processing of a form causes side effects (for example, if the form modifies a database or subscription to a service), the "post" method should be used.

### 3.1.2 <INPUT>

A text field lets the user enter a word, phrase, or series of numbers. Use the `<INPUT>` tag to place text input fields on an HTML form.

The following example creates a text element that is 25 characters long. The text field appears immediately to the right of the words "Last name:" The text field is blank when the form loads.

```html
<FORM>
  <B>Last name:</B> <INPUT TYPE="text" NAME="last_name" SIZE=25>
</FORM>
```

The output looks like:

Last name:                      

The other attributes of this tag are as follows:

- **Type**: text| password| checkbox| radio| submit| reset| file| hidden| image| button
  This attribute specifies the type of control to create. The default value for this attribute is "text".

For example:

```html
<FORM>
  <INPUT TYPE="button" VALUE="Calculate" NAME="CalcButton">
</FORM>
```

Will give an output:

![Form Example](image)

Figure 3
When a user presses the reset button, all elements in the form are reset to their default values. One can use the INPUT tag to implement reset buttons on an HTML form.

When a user presses the submit button, the form is submitted to the server specified in the form's ACTION attribute. One can use the INPUT tag to implement submit buttons on an HTML form. Figure 4.3 shows a submit button on a form. A form can have multiple submit buttons, each with different NAME and VALUE attributes. When one of the submit buttons is pressed, the form sends the server the name & value pair of that button only (in addition to the name & value pairs for all other elements on the form). Some browsers do not support multiple submit buttons on one form.

Clicking the submit button sends the form to the URL specified in the form's ACTION attribute. This action always loads a new page into the client; it might be the same as the current page, if the action specifies or is not specified.

A set of radio buttons lets the user choose one item from the set. Use the INPUT tag to implement radio buttons on an HTML form.

All radio buttons in a group have the same value for the NAME attribute. When a form is submitted to the server, the name & value pair for the radio button is sent only if the radio button is selected. For a group of radio buttons, only one name & value pair is sent to the server, because only one button in a group can be selected.

A checkbox is a toggle switch that lets the user set a value on or off. Use the INPUT tag to implement checkboxes on an HTML form. When a form is submitted to the server, the name & value pair for the checkbox is sent only if the checkbox is checked.

Multiple checkbox elements can have the same values for the NAME attribute if they have different values for the VALUE attribute. When the form is submitted, the browser sends the name & value pairs for all the checkboxes that are checked. This can yield several name & value pairs with the same name.

Image elements are graphics on an HTML form. Use the INPUT tag to implement image elements within a form. When a user clicks an image element, the form is submitted to the server specified in the form's ACTION attribute. When the form is submitted, the coordinates where the user clicked are also submitted. The coordinates are measured in pixels from the upper left corner of the image (similar to the ISMAP attribute of the IMG tag). The coordinates are sent in two name & value pairs. Appending “x” or “y” to the element’s name creates the name for each pair. For example, if the image element is named Image1 and the user clicks the image at the x coordinate 28 and the y coordinate 37, the coordinates are sent to the server using the name & value pairs Image1.x=28 and Image1.y=37.

   (i) name = cdata

This attribute assigns the control name.
(ii) \( \text{value} = \text{cdata} \)
This attribute specifies the initial value of the control. It is optional except when the type attribute has the value "radio".

(iii) \( \text{size} = \text{cdata} \)
This attribute tells the user agent the initial width of the control. The width is given in pixels except when type attribute has the value "text" or "password". In that case, its value refers to the (integer) number of characters.

(iv) \( \text{maxlength} = \text{number} \)
When the type attribute has the value "text" or "password", this attribute specifies the maximum number of characters the user may enter. This number may exceed the specified size, in which case the user agent should offer a scrolling mechanism. The default value for this attribute is an unlimited number.

**checked**

When the type attribute has the value "radio" or "checkbox", this boolean attribute specifies that the button is on. User agents must ignore this attribute for other control types.

\( \text{src} = \text{uri} \)
When the type attribute has the value "image", this attribute specifies the location of the image to be used to decorate the graphical submit button.
You can also use the INPUT tag to implement buttons on an HTML form.

### 3.1.3 <SELECT>

The SELECT element creates a menu. Each choice offered by the menu is represented by an OPTION element. A SELECT element must contain at least one OPTION element.

The attributes for the this tag are as follows:

(i) \( \text{name} = \text{cdata} \)
This attribute assigns the control name.

(ii) \( \text{size} = \text{number} \)
If a SELECT element is presented as a scrolled list box, this attribute specifies the number of rows in the list that should be visible at the same time. Visual user agents are not required to present a SELECT element as a list box; they may use any other mechanism, such as a drop-down menu.

(iii) \( \text{multiple} \)
If set, this boolean attribute allows multiple selections. If not set, the SELECT element only permits single selections.

### 3.1.4 <OPTION >

When rendering a menu choice, user agents should use the value of the label attribute of the OPTION element as the choice. If this attribute is not specified, user agents should use the contents of the OPTION element.
(i) selected
When set, this boolean attribute specifies that this option is pre-selected.

(ii) value = cdata
This attribute specifies the initial value of the control. If this attribute is not set, the initial value is set to the contents of the OPTION element.

(iii) label = text
This attribute allows authors to specify a shorter label for an option than the content of the OPTION element. When specified, user agents should use the value of this attribute rather than the content of the OPTION element as the option label.

The use of the select and the option tag are explained well by the following example:

```html
<HTML>
<FORM name="test" action="forms.html" method="post">
Name:
<INPUT type="text" id="name" value="Dr. Anurag Verma" size=25 >
<br>
<SELECT name="age" value="24">1,2,3,4,5,6,7,8,9,10
   <OPTION>1</OPTION>
   <OPTION>2</OPTION>
   <OPTION selected>3</OPTION>
   <OPTION>4</OPTION>
   <OPTION>5</OPTION>
   <OPTION>6</OPTION>
   <OPTION>7</OPTION>
</SELECT>
<INPUT type="submit" name="submit" value="submit">
</FORM>
</HTML>
```

This code will give a form that looks like the explorer screen shown below:

![Figure 4](image-url)
3.1.5 <TEXTAREA>

The TEXTAREA element creates a multi-line text input control. User agents should use the contents of this element as the initial value of the control and should render this text initially.

The attributes for the tag are as follows:

(i)  name = cdata
     This attribute assigns the control name.

(ii) rows = number
     This attribute specifies the number of visible text lines. Users should be able to enter more lines than this, so user agents should provide some means to scroll through the contents of the control when the contents extend beyond the visible area.

(iii) cols = number
     This attribute specifies the visible width in average character widths. Users should be able to enter longer lines than this, so user agents should provide some means to scroll through the contents of the control when the contents extend beyond the visible area. User agents may wrap visible text lines to keep long lines visible without the need for scrolling.

The following HTML example displays two text elements, a select element, and three radio buttons, all of which have default values. The form also has a reset button named "Defaults." If the user changes the value of any of the elements and then clicks the Defaults button, the original values are restored.

<HTML>
<FORM NAME="form1">
<br>
<B>City: </B>
<INPUT TYPE="text" NAME="city" VALUE="Santa Cruz" SIZE="20">
<B>State: </B>
<INPUT TYPE="text" NAME="state" VALUE="CA" SIZE="2">
<P><SELECT NAME="colorChoice">
   <OPTION SELECTED> Blue
   <OPTION> Yellow
   <OPTION> Green
   <OPTION> Red
</SELECT>
<P><INPUT TYPE="radio" NAME="musicChoice" VALUE="soul-and-r&b" CHECKED> Soul and R&B
<BR><INPUT TYPE="radio" NAME="musicChoice" VALUE="jazz"> Jazz
<BR><INPUT TYPE="radio" NAME="musicChoice" VALUE="classical"> Classical
</FORM></HTML>
renders the Explorer to look like this

![Image of HTML form rendered in Internet Explorer]

**Figure 5**

**Note:** You can also include a form in a document that contains a body part.

**Hidden elements**

Hidden elements are text elements that don't display on the form. Use the INPUT tag to implement hidden elements. A hidden element is used for passing information to the server when a form is submitted. A hidden element cannot be seen or modified by a user (other than by viewing the source of the HTML), but by using JavaScript, you can programmatically change its value. You can use hidden elements for client/server communication or to pass state information from one script or form to another.

When a form is submitted to the server, a hidden element's name & value pair is always sent.
The following example creates a form called LoginForm that contains text fields for user name and password, a submit button, and a cancel button. A hidden element, DefaultPass, stores the initial value of the password field.

```
<FORM NAME="LoginForm">
  <B>User name:</B>
  <INPUT TYPE="text" NAME="userName" SIZE="10">
  <P>
  <B>Password:</B>
  <INPUT TYPE="password" NAME="password" SIZE=12 VALUE="treasure">
  <INPUT TYPE="hidden" NAME="DefaultPass" VALUE="treasure">
  <P>
  <INPUT TYPE="submit" VALUE="Log in">
  <INPUT TYPE="button" VALUE="Cancel" onClick="window.close()">
</FORM>
```

### 3.2 Password elements

Password elements are text-input fields on an HTML form that conceal their value by displaying asterisks (*). When the user enters text into the field, asterisks (*) hide anything entered from view.

Even though passwords are masked onscreen, the password is sent to the server as straight text and is not encrypted when the form is submitted, unless the server itself uses an encryption method. Be cautious when using password fields because they could be intercepted and read by anyone. The following HTML example creates a form in which if the user enters a password containing more than 25 characters, the text scrolls to make room for the additional characters.

```
<FORM>
  <B>User name:</B>
  <INPUT TYPE="text" NAME="username" SIZE=10>
  <B>Password:</B>
  <INPUT TYPE="password" NAME="password" VALUE="" SIZE=25>
</FORM>
```

### 3.3 Sending a form data to a server

HTML forms collect data, but they do not usually process it. To process a form, you can submit the form to a program stored on a web server. The form and the server-side program should be designed together so that the program can process the form data being sent. When a form is submitted, each form element defined with INPUT, SELECT, or TEXTAREA tag is sent to the server in the format name & value; this is called a name & value pair. The name comes from the tag's NAME attribute, and the value is the value of the form element when submitted.
For example, the following form contains an INPUT tag that defines a text element called LastName.

```html
<Form>
  <B>Last name:</B>
  <Input type="text" name="LastName" value="">
  <Input type="submit" name="SubmitButton" value="Done">
</Form>
```

Suppose the user enters the value "Mitchell" for LastName and presses the Done button; the form is submitted and the name & value pair LastName=Mitchell is sent to the server.

**HTML: PRACTICAL**

**EXERCISE:** Design a web page depicted below?

To design this web page you need to develop three html files namely Index.htm, top1.html, home.htm. Index.htm defines the frameset and the two frames in which the two files top1.html, home.htm are called. File top1.html displays the title and the hyper linked menu of the web page in MyFrameOne whereas the file home.htm displays the tabular information of the web page in MyFrameTwo.

The source codes for developing these html files are given below.
File1: Index.htm:

```html
<HTML>
<TITLE>CAS Training</TITLE>
<frameset rows="152,*" rightmargin=0 framespacing=0 STYLE="width:0px;PADDING:0px;MARGIN:0px;margin-right:0px"
frameborder='0'topmargin='0' >
    <frame src="top1.html" scrolling=no rightmargin=0 name="MyFrameOne">
    <frame src="/home.htm" rightmargin=0 frameborder='0' name="MyFrameTwo">
</frameset>
</HTML>
```

File2: top1.html

```html
<html>
<head>
    <title>Your Company Name</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <link rel="stylesheet" href="css/styles.css" type="text/css">
</head>
<body bgcolor="#999999" leftmargin="0" topmargin="0" marginwidth="0"
marginheight="0">
    <table cellspacing="0" cellpadding="0" border="0" align="center" height="115" width="100%">
        <tr>
            <td style="padding-left:25px" background="images/main1.jpg" class="logo">
                Design and Development of Web based Application using .NET technology</td>
        </tr>
    </table>
    <table cellspacing="0" cellpadding="0" border="0" width="100%">
        <tr bgcolor="#99cccc">
            <td height="32" bgcolor="#99cccc" width="100%">
                <table cellspacing="0" cellpadding="0" border="0">
                    <tr bgcolor="#99cccc">
                        <td colspan="2" bgcolor="#99cccc">
                            <table cellspacing="0" cellpadding="0" border="0">
                                <tr height="32" bgcolor="#99cccc" class="text" align="left" fontsize="2">
                                    <a href="home.htm" target="MyFrameTwo">Home</a> | <a href="Introduction.htm" target="MyFrameTwo">Introduction</a> |
                                </tr>
                            </table>
                        </td>
                    </tr>
                </table>
            </td>
        </tr>
    </table>
</body>
</html>
```
Content Designing using HTML (Frames and Forms)

120

File3: home.htm

<html>
<head>
<title>Training Program under Center of Advanced Studies</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<style>
!-->
.MsoNormal {h5 {p.MsoHeading9, li.MsoHeading9, div.MsoHeading9 {span.SpellE {}}}
-section1 {page:Section1;}
--> </style>
</head>
<body bgcolor="#e4ffff" class="Normal" lang="EN-US" topmargin=0>
<div class="Section1">
<div align="center">
<table class="MsoTableGrid" border=".5" cellspacing="5" cellpadding="5">
<tr>
<td width="590" height="30" colspan="2" valign="top" class="Normal">

</td>
</tr>
</table>
</div>
</div>
</body>
</html>
<p>Training Program under Center of Advanced Studies</p>

<table>
<thead>
<tr>
<th>Title</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design &amp; Development of Web based Application using .NET Technology</td>
<td>22&lt;sup&gt;nd&lt;/sup&gt; November to 12&lt;sup&gt;th&lt;/sup&gt; December’ 2006</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sponsoring authority</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Training Programme under CAFT “Online Content Creation and Management in an eLearning Environment”
**Indian Council of Agricultural Research**

**Course Director** Alka Arora

<table>
<thead>
<tr>
<th>Number of participants</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last date for nomination</td>
<td>10th Oct. 2006</td>
</tr>
</tbody>
</table>

*Training Programme under CAFT “Online Content Creation and Management in an eLearning Environment”*
Selected candidates informed

Venue  

Division of Computer Applications  
Indian Agricultural Statistics Research Institute (ICAR)  
Library Avenue, Pusa New Delhi -110 012