LEARNER MANUAL

APPLY ADVANCED HTML AND ASSOCIATED TECHNIQUES TO BUILD A WEBSITE FOR BUSINESS APPLICATIONS

US ID: 115368

NQF LEVEL: 5

CREDITS: 12

NOTIONAL HOURS: 120

Contents

This manual was compiled by SM Support on behalf of the training provider.

HOW TO USE THIS GUIDE

This workbook belongs to you. It is designed to serve as a guide for the duration of your training programme. It contains readings, activities, and application aids that will assist you in developing the knowledge and skills stipulated in the specific outcomes and assessment criteria. Follow along in the guide as the facilitator takes you through the material, and feel free to make notes and diagrams that will help you to clarify or retain information. Jot down things that work well or ideas that come from the group. Also, note any points you would like to explore further. Participate actively in the skill practice activities, as they will give you an opportunity to gain insights from other people's experiences and to practice the skills. Do not forget to share your own experiences so that others can learn from you too.

ICONS

HOW YOU WILL LEARN

The programme methodology includes facilitator presentations, readings, individual activities, group discussions, and skill application exercises.

PROGRAMME OVERVIEW

PURPOSE

People credited with this unit standard are able to:

- Describe the use of HTML editors and other web site design/ maintenance tools
- Demonstrate an understanding of web browser plug-ins

LEARNING ASSUMPTIONS

The credit value of this unit is based on a person having prior knowledge and skills to:

- Demonstrate PC competency skills (End-User Computing unit Standards, at least up to NQF level 3.)
- Demonstrate an understanding of the use of web-sites in Business

HOW YOU WILL LEARN

The programme methodology includes facilitator presentations, readings, individual activities, group discussions, and skill application exercises.

HOW YOU WILL BE ASSESSED

This programme has been aligned to registered unit standards. You will be assessed against the outcomes of the unit standards by completing a knowledge assignment that covers the essential embedded knowledge stipulated in the unit standards. When you are assessed as competent against the unit standards, you will receive a certificate of competence and be awarded 3 credits towards a National Qualification.

SESSION 1: WORKING WITH ADVANCED HTML FEATURES

On completion of this section you will be able to discuss the need for advanced HTML features.

1. The discussion identifies advanced HTML features
2. The discussion demonstrates the use of the features identified
3. The discussion demonstrates the application of simple Cascading Style Sheets (CSS)
4. Demonstrate the use of Dynamic HTML and XML
5. The demonstration defines Dynamic HTML concepts
6. The demonstration defines XML concepts
7. The demonstration illustrates the use of Dynamic HTML and XML in a business context

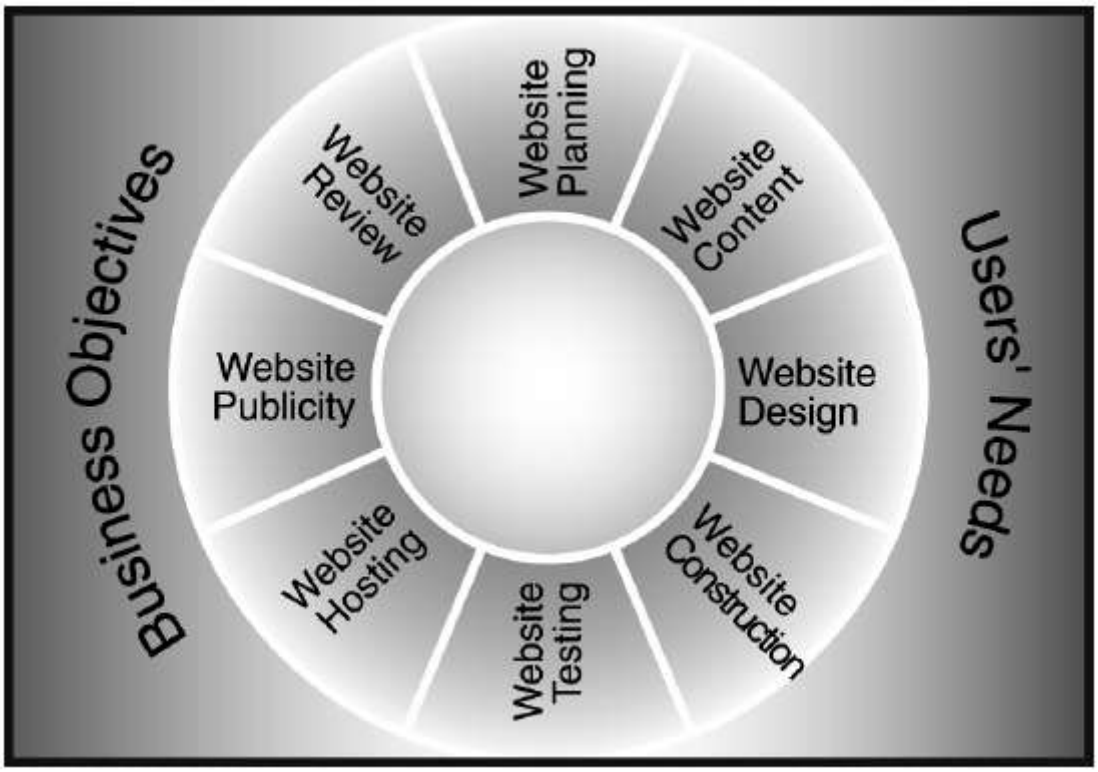
1.1 INTRODUCTION

Website Development is a process for creating a new website or implementing changes to one already in use, e.g. adding a significant new section to a live site.
In simple terms, the process represents a framework within which all activities from inception to review (and eventual demise, if necessary) can take place.

There are 8 steps in the development process. These are:

- **Planning**: Decide why you want a website and what to create.
- **Content**: Create a list of the content you want.
- **Design**: Create a design for displaying the content.
- **Construction**: Write the code and load up your content.
- **Test**: Make sure everything works properly.
- **Hosting**: Choose a domain name & find a place to put your site on the internet.
- **Publicity**: Build traffic via publicity the site.
- Review: Review the site at intervals it to make sure it succeeds.

Of course, Website Development does not happen just for fun it must be initiated in some way.



As can be seen above, the development process takes place within the bounds of Business Goals and User Needs. It is these that initiate and guide the course of planning, design, content, etc. Until you explore your goals and users, your website simply has no reason to exist.

1.2 UNDERSTANDING BASIC HTML WEBSITES

HTML stands for HyperText Markup Language. Just as we, humans, use languages to communicate with each other, HTML is a language of the internet. Web browsers like Google Chrome read and interpret HTML to render the information you request for on the web. HTML is essentially the building block of every website. It defines the structure of every web page and its content.

1.2.1 COMMON HTML TERMS

While getting started with HTML, it's important to understand a few common terms associated with the language:

Elements: These are designators that define the order and structure of the content within a web page. , <p> , , <h1>, <title> etc. are all examples of basic HTML elements. Each HTML element is enclosed within a start and end tag as shown below.

 …

Tags: The use of angle brackets around an HTML element creates what's called an HTML tag. HTML tags most often occur in pairs of start and end tags.

A start tag indicates the beginning of an HTML element and looks like this:

An end tag indicates the end of an HTML element and consists of a forward slash as shown below:

/

Attributes: We use attributes to provide additional information about an HTML element. Some common attributes include id, class, src and href. Attributes are defined within the start tag of an element after its name. Each attribute is defined by a name and a value. For example, the <a> element below includes a href attribute as shown below:

 Google Search Engine

1.2.3 WHAT IS CSS?

CSS stands for Cascading Style Sheets. CSS defines the visual style and appearance of the web pages. So if you see italicized text or a background image on a web page, it is CSS that is doing the trick!

COMMON CSS TERMS

Here are some of the common CSS terms that are useful to know:

```
a {
color: #484848;
font-size: 14px;
font-weight: 300px;
 }
```

Selectors: A selector designates the exact HTML elements to target for styling. Selector generally targets all HTML elements of one type directly or a specific instance of an element, identified by an attribute such as id or class. The styles to be applied to the targeted element are enclosed within curly brackets following the selectors. For example, the selector in the above code is targeting all the <a> elements.

Properties: A property determines the styles that will be applied to an HTML element. There are several properties that are available for use. Some of the common properties are font, background-color, size and color. In the example above, we are using the color, font-size and font-weight properties to style the <a> element.

Values: Each property has a value associated with it. The value defines the behavior of the property. For example, the value 14px defines the behavior of the font-size property.

DIFFERENCE BETWEEN HTML & CSS?

Even though very often HTML & CSS are used together in the same context, they are far from being the same. While HTML defines the structure of the web page, CSS defines the style and appearance.

To make this easier to understand, let's compare a website to a house. HTML is like the bricks that make up the house while CSS is the paint and decor that makes the house presentable and beautiful.

A webpage can exist without CSS, but never without HTML.

1.2.4 DEVELOP A SIMPLE WEB PAGE USING HTML & CSS

Now let's build a simple HTML web page. We will need:

- A Text Editor to write the code for your web page.
- A Web Browser to view your web page. Anything from Google Chrome to internet explorer will do.

Step 1

Create a folder and name it. We are going with MyWebPage1.0.

Step 2

Create a text file inside the folder and paste the following HTML code in it. Name it as Webpage.html and save as an HTML file type.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>My First Web Page</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <h1> Coding is so much fun!</h1>
    <p> The best way to learn coding is to explore and experiment with the
different styles, attributes and tags. Have fun with it!</p>
  </body>
</html>
```

Step 3

Next, create another text file inside the folder and paste the following CSS code in it. Name it as style.css and save it as a CSS file type.

```css
/* This is the css code for styling the HTML page */

html, body, div, span{
  margin: 0;
  padding: 0;
  border: 0;
  font: inherit;
  vertical-align: baseline;
}

body{
  background-color: #f6f6f6;
  font-size:14px;
}

h1{
  color: #f6a2a2;
  font-family: Georgia, sans-serif;
  font-size:34px;
  font-weight: bold;
  text-align: center;
  letter-spacing: 3px;
  padding: 50px;
  text-transform:uppercase;
}

p{
  text-align: center;
  font-family: Times new roman, sans-serif;
  font-size:22px;
  letter-spacing: 1px;
  font-style:italic;
}

a{
  color: #e4d51e;
}
```

That’s it, now double click on your HTML file to run it on your web browser. Here’s a screenshot of the webpage you can expect to see on your browser.

And there you go, you've created your first web page! Feel free to play around by changing the HTML and CSS code to alter your web page. The best way to learn HTML and CSS is to really just experiment and have fun exploring the various possibilities that they offer.

Example

Take the following (incomplete) sample HTML page as a starting point.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
</head>
<body>
<p>
This is a sample web page from thesitewizard.com's HTML
tutorial.
</p>
</body>
</html>
```

Before we discuss the code above, please copy it into your text editor, and save it as a file named "sample.html" on your desktop. I know it's easy to just dismiss my words here and skip this step, but doing it will speed up your learning and improve your understanding of what I'm talking about later. (You'll also be modifying the code as we go along.) There's also a secondary (less important) benefit in that it'll help you get familiar with your text editor.

For those who don't know how to do what I just said in the paragraph above, click once in the box above containing the code. The contents of the box should automatically be highlighted. If it isn't, drag your mouse over the text and manually select everything. Then click your right mouse button. In the menu that appears, click the "Copy" item. Start up your text editor. Click "Edit" on the menu bar followed by the "Paste" item in the drop-down menu that appears. The text you see in the box above should appear in your text editor.

Depending on which editor you've installed, the text above may appear in a variety of colours. Don't worry about that. The colours are what programmers call "syntax highlighting". Just ignore it. (And don't worry if you don't see those colours either. They are not important.)

To save the file, click the "File" menu, followed by the "Save" item in the submenu that appears. When prompted for a filename, select the appropriate folder for the file to be saved (for example, your desktop, so that you won't forget where you put it) and type "sample.html" (without the quotes). Note that I'm assuming that you're not using Notepad. If you use Notepad, this won't work, since Notepad will invisibly change your filename to "sample.html.txt" behind your back. Don't use Notepad.

Now open the file in your web browser. One way to do it is to doubleclick the file on your desktop. The file should appear in your default web browser. Keep both the browser and the text editor open. That is, don't close or exit either program.

The Logic of HTML

Look at the HTML code (either in your text editor or by scrolling up in this article).

Notice that the words displayed in your web browser, "This is a sample web page from the sitewizard.com's HTML tutorial" is sandwiched between " " and " ". This entire block is itself sandwiched between "<body>" and "</body>". In fact, if you're observant, you'll notice that there are other pairs of words enclosed in angle brackets like the "<head>" and "</head>" pair, as well as the "<html>" and "</html>" pair.

All these angle-bracketed words (or letters) are commonly called "**HTML tags**". In HTML, the majority of tags come in pairs, with the unadorned version, like "<body>", being used to indicate the start of something, and the version with a preceding slash ("/"), like "</body>", being used to flag the end of something.

Each tag has a specific function. For example, the " " and " " tag pair is used to indicate the start and end of a paragraph. (Think of the "p" in " " as referring to "paragraph".)

The Structure of an HTML Document

Before we discuss the structure of a web page, consider a normal business letter in the brick and mortar world. Such letters actually have a particular form: if the paper you're using does not have a letterhead, you will normally have to type (or write) your own address at the top of the page, followed by the address of the person you're writing to below. Only then do you begin your letter proper with something like "Dear...". There is typically a subject line following that,

and below that comes the body of your letter where you finally get to say the things you wanted to say.

Web pages also have a particular format.

1. The first line of the page identifies the type and version of HTML that your web page is using.
2. The rest of the page is enclosed between the "<html>" / "</html>" tag pair. Code enclosed between these two tags form the web page proper.
3. The first part of the web page is what is known as the HEAD section. The section begins with "<head>" and ends with "</head>". It contains information meant for web browsers and search engines. Code placed in this section is not displayed in a web browser. It's like the addresses you place at the beginning of a business letter — they serve a purpose, but they're not actually part of the main content. Anyway, I'm sure you noticed that the HEAD section in my example code is empty at the moment. That's because I didn't want to confuse you by cluttering the file with too many things. We'll insert something into this section later in this chapter.
4. The part of the web page that contains the information displayed in a web browser is called the BODY section, and it is, as you've probably guessed by now, enclosed between the "<body>" and "</body>" tag pair.

Each of these things is discussed in detail below.

DOCTYPE or DTD

In the example code you used above, the first line reads as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
```

Unlike the other lines on your web page, this first line is not an HTML tag. As mentioned above, it's a sort of version identifier that tells web browsers that your page is using version 4.01 of the HTML specifications. At the time I write this, as far as I can tell, the most commonly used versions of HTML on the Internet are HTML 4 (or 4.01 to be exact) and XHTML 1.0. For all practical intents and purposes, these two versions are actually mostly the same, except that XHTML is fussier about some things, like requiring small letters (lowercase) words for your tags, requiring closing tags for everything, and some other stuff.

We will only be dealing with HTML 4.01 in this tutorial. Don't worry. You don't lose out anything by using HTML 4 as compared to XHTML. In spite of its flashier name (with the extra letter), XHTML doesn't give you more features to use on your web page. It mostly just makes the

browser fussier about what it will accept from your web page. It also adds complications with cross-browser compatibility (mainly with old versions of Internet Explorer) that will make your life a bit more miserable if you're a purist and insist on strict compliance with every aspect of XHTML. Since you're just starting out, it's easiest to just begin with HTML 4. (And there's no real need to transition to XHTML either. HTML 4 is complete in itself.)

The version line is commonly called a **DOCTYPE** by webmasters (since the line begins with the word DOCTYPE). Officially, it is known as a **Document Type Definition** (or "**DTD**" for those who love abbreviations). As its name indicates, it does more than tell web browsers the version of HTML you're using. It also indicates whether your web page is one of 3 types: whether it uses frames (a type of HTML technology), whether it is a normal web page ("transitional"), or whether it is a normal web page that uses a limited subset of HTML ("strict").

For those curious about what the DOCTYPEs look like for the other document types, they are as follows:

HTML 4.01 Strict DOCTYPE

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
```

HTML 4.01 Frameset DOCTYPE

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">
```

Anyway, there's no need to memorize any of these DOCTYPEs. Just copy and paste them into your web page as needed.

The HEAD Section: Introducing The TITLE Tag

One of the most important things that belong in the HEAD section is the TITLE tag for your web page. This tag provides web browsers and search engines the title of your page. The title you give in this tag doesn't actually appear on your web page itself, but it is placed in the title bar of the web browser window when it displays your page. To understand what I mean by this, glance at the top of your browser window for now. Do NOT scroll the page to the top or anything like that. Just look upwards at the top edge of the browser. In most web browsers, you should be able to see the words "HTML Tutorial - Learn to Design a Website using HTML (thesitewizard.com)" somewhere in the title bar. Some browsers put these words on the tab for the current window. This is the title that I set for this article.

Switch to your text editor and add the following line into your HEAD section.

```
<title>My First Web Page</title>
```

For those not sure what I mean, move your cursor to the end of the line containing the <head> tag. Hit the ENTER key to create a new line underneath. Then type in the words I gave in the block above. Actually, you don't really have to use the words "My First Web Page". Use whatever words you want as the title. But make sure those words are enclosed between the opening <title> tag and the closing </title> tag. And ensure that this entire sequence occurs before the closing </head> tag. In other words, when I say to add a certain line into the HEAD section, it means that you have to add that line between the opening <head> tag and the corresponding closing </head> tag.

Save your page and reload (refresh) it in your web browser. For those who are not sure how to reload a page in your browser, find out how. In most browsers, hitting Ctrl+R should do the trick (that is, holding down the Ctrl key while typing the letter "r" on the keyboard). In addition, if you use Windows, a quick way to save your work in most (if not all) text editors is to hit Ctrl+S, that is, hold down the Ctrl key and type "s" on the keyboard. (I'm not sure if this works across the board on Mac software too.)

Now look at the title bar of the browser window. You should be able to see the title you set. Notice that the words you typed do not appear in the main body of your web page itself. The <title> only sets the title of your web page for the benefit of web browsers and search engines. It does not change your web page's appearance. Anything that you want displayed on your web page itself has to be inserted into the BODY section.

Again, please do not quit your text editor or web browser.

The BODY Section and The Paragraph Tag

Although I'll introduce the majority of the HTML tags dealing with text in the next chapter, we will handle the paragraph tag, , here. This will also allow me to demonstrate one very important feature of HTML, namely, whitespace characters do not matter in the portion of the web page between most opening and closing HTML tags. (Don't worry if the last part of that sentence doesn't make sense. That's what this whole section is here for: to explain it.)

Before we get ahead of ourselves, let me first state that the tag signals to the browser that it is to display what follows as a single paragraph until it reaches a tag.

In your text editor, put your text cursor at the end of the line that says "This is a sample web page from thesitewizard.com's HTML tutorial." That is, move your cursor until it is just after the

full stop (period) in that sentence. Hit the ENTER key (or RETURN key if you use a Mac). This creates a new line in the text editor. Now type some other sentence into that new blank line. For those who don't know what to say, type "TheSiteWizard.com is a useful resource for webmasters" (without the quotes).

Save the page and reload it in your browser, and look carefully at the placement of your new sentence. Notice that even though you put the sentence on a new line in your text editor, your browser displayed the sentence immediately following the previous one, so as to make both sentences part of the same paragraph.

Return to your text editor and put your cursor between the words "from" and "thesitewizard.com's". At this time, there is only one blank space between those two words. Add some more spaces by hitting your space bar several times. It doesn't matter how many spaces you add, as long as it's a significant number so that you can easily spot that there's a large unnatural gap between those two words. Hit the space bar 10 times if you can't decide.

Once again, save your page and refresh it in the browser. Look at your paragraph in the browser again. Observe that your paragraph looks no different from before. It's as though the web browser swallowed all your extra spaces.

The space character and the new line character (the invisible character or sequence of characters that the text editor inserts into your web page when you hit the ENTER or RETURN key) are called "whitespace" characters. Multiple whitespace characters are replaced by a single whitespace character by all web browsers. In HTML, you do not format a web page by adding spaces or new lines. It won't work. As you have noticed from above, the browser just ignores the extra spaces or lines you add. Instead, you format a page in HTML by adding different HTML tags, or by using CSS.

How does one add a new paragraph to a web page? I'm sure at this point, most of you have already (long) guessed the answer. But let's try it out so that you can see it in action. (I also want to use it to demonstrate something else about HTML. So please do it even though it seems really obvious.)

Move your cursor to the end of the </p> line. Hit the ENTER key. Then insert the following into the new line that you created.

```
<p>This is a new paragraph because I used a new paragraph tag to start it off.</p>
```

Save the page and reload it in your browser. The newest addition to your web page should now appear as a new paragraph in your browser.

Switch back to your text editor again. Notice that unlike the previous paragraph, I didn't put my <p> tag on its own line. Instead, I just typed the sentence belonging to the paragraph immediately after the tag, and yet everything worked out the same as when I put the tag on a separate line. Whether or not there is any whitespace between the paragraph tag and my actual text doesn't matter at all.

We recommend that you play around with this a while to get familiar with how the web browsers interpret white space and the paragraph tags. That is, add new paragraphs, or new sentences to your existing paragraph tags, put all sentences and tags on a single line, and reload the page in your browser to see its effects.

Recommendations (to Protect Your Sanity)

From the above experiments, you have probably realised that you can probably put your entire web page on a single line in your text editor, and everything will still show up correctly in your web browser.

While that is true, it's best that you don't do that. Although web browsers will have no trouble dealing with that kind of code, you, the webmaster maintaining the page, will. With everything compressed to occupy minimum space, it's very easy for you to miss something and create errors on your web page. The minimal space saving you get from doing those kinds of silly tricks is just not worth it.

1.3 ADVANCED HTML FEATURES

In this section we shall discuss the advanced HTML features.

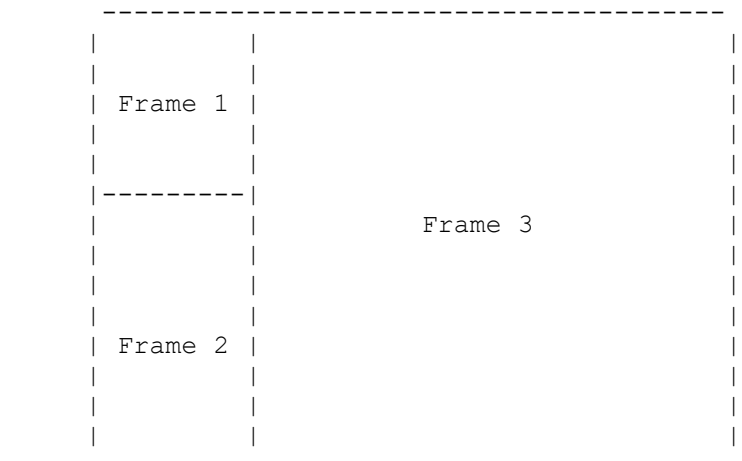
1.3.1 FRAMES

HTML frames allow authors to present documents in multiple views, which may be independent windows or subwindows. Multiple views offer designers a way to keep certain information visible, while other views are scrolled or replaced. For example, within the same window, one frame might display a static banner, a second a navigation menu, and a third the main document that can be scrolled through or replaced by navigating in the second frame.

Here is a simple frame document:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
   "http://www.w3.org/TR/html4/frameset.dtd">
<HTML>
<HEAD>
<TITLE>A simple frameset document</TITLE>
</HEAD>
<FRAMESET cols="20%, 80%">
  <FRAMESET rows="100, 200">
      <FRAME src="contents_of_frame1.html">
      <FRAME src="contents_of_frame2.gif">
  </FRAMESET>
  <FRAME src="contents_of_frame3.html">
  <NOFRAMES>
      <P>This frameset document contains:
      <UL>
         <LI><A href="contents_of_frame1.html">Some neat contents</A>
         <LI><IMG src="contents_of_frame2.gif" alt="A neat image">
         <LI><A href="contents_of_frame3.html">Some other neat
contents</A>
      </UL>
  </NOFRAMES>
</FRAMESET>
</HTML>
```

That might create a frame layout something like this:



If the user agent can't display frames or is configured not to, it will render the contents of the NOFRAMES element.

Layout of frames

An HTML document that describes frame layout (called a frameset document) has a different makeup than an HTML document without frames. A standard document has one HEAD section and one BODY. A frameset 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 FRAMESET element

```
<![ %HTML.Frameset; [
<!ELEMENT FRAMESET - - ((FRAMESET|FRAME)+ & NOFRAMES?) -- window
subdivision-->
<!ATTLIST FRAMESET
  %coreattrs;                          -- id, class, style, title --
  rows        %MultiLengths; #IMPLIED  -- list of lengths,
                                          default: 100% (1 row) --
  cols        %MultiLengths; #IMPLIED  -- list of lengths,
                                          default: 100% (1 col) --
  onload      %Script;       #IMPLIED  -- all the frames have been loaded
--
  onunload    %Script;       #IMPLIED  -- all the frames have been removed
--
  >
]]>
```

Attribute definitions

Rows = *multi-length-list* [CN]

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.

Cols = *multi-length-list* [CN]

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.

Attributes defined elsewhere

- id, class (document-wide identifiers)
- title (element title)
- style (inline style information)
- onload, onunload (intrinsic events)

The FRAMESET element specifies the layout of the main user window in terms of rectangular subspaces.

Rows and columns

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>
```

Absolute lengths that do not sum to 100% of the real available space should be adjusted by the user agent. 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.

Nested frame sets

Framesets may be nested to any level.

In the following example, the outer `FRAMESET` divides the available space into three equal columns. The inner `FRAMESET` then divides the second area into two rows of unequal height.

```
<FRAMESET cols="33%, 33%, 34%">
     ...contents of first frame...
     <FRAMESET rows="40%, 50%">
        ...contents of second frame, first row...
        ...contents of second frame, second row...
     </FRAMESET>
     ...contents of third frame...
</FRAMESET>
```

Sharing data among frames

Authors may share data among several frames by including this data via an OBJECT element. Authors should include the OBJECT element in the HEAD element of a frameset document and name it with the id attribute. Any document that is the contents of a frame in the frameset may refer to this identifier.

The following example illustrates how a script might refer to an OBJECT element defined for an entire frameset:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
   "http://www.w3.org/TR/html4/frameset.dtd">
<HTML>
<HEAD>
<TITLE>This is a frameset with OBJECT in the HEAD</TITLE>
<!-- This OBJECT is not rendered! -->
<OBJECT id="myobject" data="data.bar"></OBJECT>
</HEAD>
<FRAMESET>
    <FRAME src="bianca.html" name="bianca">
</FRAMESET>
</HTML>
```

```
  <!-- In bianca.html -->
  <HTML>
  <HEAD>
  <TITLE>Bianca's page</TITLE>
  </HEAD>
  <BODY>
  ...the beginning of the document...
  <P>
  <SCRIPT type="text/javascript">
  parent.myobject.myproperty
  </SCRIPT>
  ...the rest of the document...
  </BODY>
  </HTML>
```

16.2.2 The FRAME element

```
<![ %HTML.Frameset; [
<!-- reserved frame names start with "_" otherwise starts with letter -->
<!ELEMENT FRAME - O EMPTY              -- subwindow -->
<!ATTLIST FRAME
  %coreattrs;                          -- id, class, style, title --
  longdesc    %URI;          #IMPLIED  -- link to long description
                                          (complements title) --
  name        CDATA          #IMPLIED  -- name of frame for targetting --
  src         %URI;          #IMPLIED  -- source of frame content --
  frameborder (1|0)          1         -- request frame borders? --
  marginwidth %Pixels;       #IMPLIED  -- margin widths in pixels --
  marginheight %Pixels;      #IMPLIED  -- margin height in pixels --
  noresize    (noresize)     #IMPLIED  -- allow users to resize frames? --
  scrolling   (yes|no|auto)  auto      -- scrollbar or none --
  >
]]>
```

Attribute definitions

name = *cdata* [CI]

This attribute assigns a name to the current frame. This name may be used as the target of subsequent links.

longdesc = *uri* [CT]

This attribute specifies a link to a long description of the frame. This description should supplement the short description provided using the title attribute, and may be particularly useful for non-visual user agents.

src = *uri* [CT]

This attribute specifies the location of the initial contents to be contained in the frame.

noresize [CI]

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

scrolling = auto|yes|no [CI]

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.

frameborder = 1|0 [CN]

This attribute provides the user agent with information about the frame border.
Possible values:

- 1: This value tells the user agent to draw a separator between this frame and every adjoining frame. This is the default value.
- 0: This value tells the user agent not to draw a separator between this frame and every adjoining frame. Note that separators may be drawn next to this frame nonetheless if specified by other frames.

marginwidth = *pixels* [CN]

This attribute specifies the amount of space to be left between the frame's contents in its left and right margins. The value must be greater than zero (pixels). The default value depends on the user agent.

marginheight = *pixels* [CN]

This attribute specifies the amount of space to be left between the frame's contents in its top and bottom margins. The value must be greater than zero (pixels). The default value depends on the user agent.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `title` (element title)
- `style` (inline style information)

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

Setting the initial contents of a frame

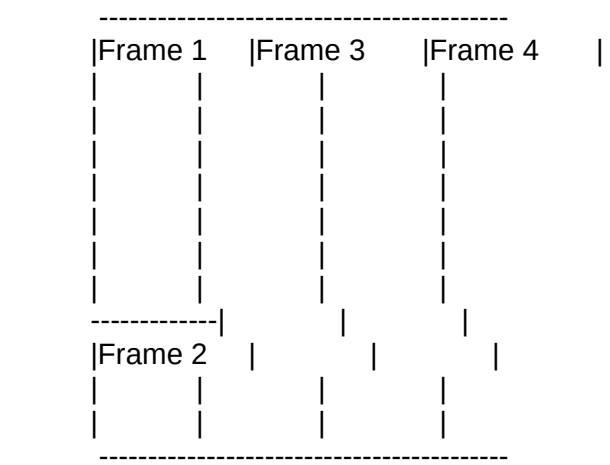
The src attribute specifies the initial document the frame will contain.

The following example HTML document:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
   "http://www.w3.org/TR/html4/frameset.dtd">
<HTML>
<HEAD>
<TITLE>A frameset document</TITLE>
</HEAD>
<FRAMESET cols="33%,33%,33%">
  <FRAMESET rows="*,200">
      <FRAME src="contents_of_frame1.html">
```

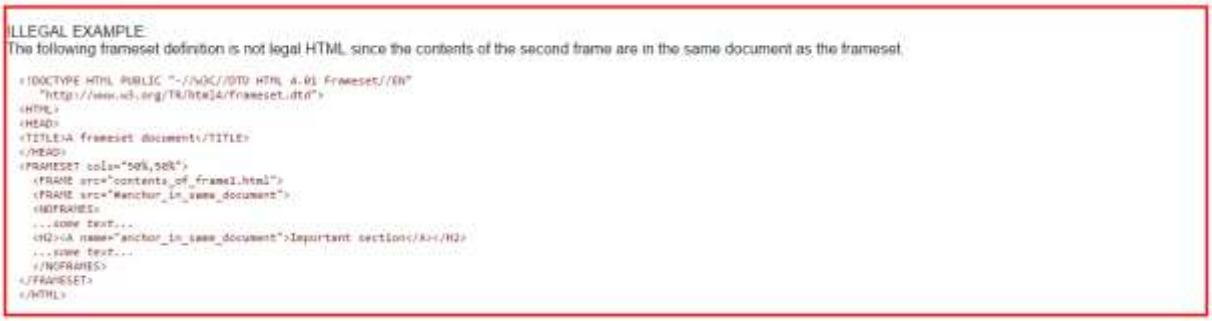
```
        <FRAME src="contents_of_frame2.gif">
    </FRAMESET>
    <FRAME src="contents_of_frame3.html">
    <FRAME src="contents_of_frame4.html">
  </FRAMESET>
  </HTML>
```

Should create a frame layout something like this:



and cause the user agent to load each file into a separate view.

The contents of a frame must not be in the same document as the frame's definition.



Visual rendering of a frame

The following example illustrates the usage of the decorative FRAME attributes. We specify that frame 1 will allow no scroll bars. Frame 2 will leave white space around its contents (initially, an image file) and the frame will not be resizeable. No border will be drawn between frames 3 and 4. Borders will be drawn (by default) between frames 1, 2, and 3.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
   "http://www.w3.org/TR/html4/frameset.dtd">
<HTML>
<HEAD>
<TITLE>A frameset document</TITLE>
</HEAD>
<FRAMESET cols="33%,33%,33%">
  <FRAMESET rows="*,200">
      <FRAME src="contents_of_frame1.html" scrolling="no">
      <FRAME src="contents_of_frame2.gif"
```

```
                marginwidth="10" marginheight="15"
                noresize>
  </FRAMESET>
  <FRAME src="contents_of_frame3.html" frameborder="0">
  <FRAME src="contents_of_frame4.html" frameborder="0">
</FRAMESET>
</HTML>
```

1.3.2 FORMS

If you've been on the Internet for a while, you've probably filled out a number of online forms. Forms are used to obtain information from your visitors right through your website. Your visitors can input their information into your form, click on a "submit" button and their information will be directed to a location you specify.
If you're running a business on the Internet, using a form to process your customer's orders is an absolute must. I'm amazed at the number of websites that are still processing their customer's orders via snail mail. If you're not automating your ordering process, you're losing a significant amount of business every day.
Most Internet users won't take the time to print out an order form, place it in an envelope and send you their order -- their time is valuable. You must make the ordering process as simple as possible. This includes setting up a form on your website to process their orders electronically.

Forms are used for all of the following:

- Order forms
- Email subscriptions
- Contest registrations
- Databases
- Auto responders
- User identifications and passwords
- Feedback

BASIC FORM TUTORIAL

Your first step in creating a form will be to get a good form script. This script will reside on your server within your CGI-bin and will be used to process your form's information. You can find some great scripts here: http://cgi.resourceindex.com/Programs_and_Scripts/Perl/

To insert a form on your web page, we will begin with <FORM> and end with </FORM>. All of the FORM elements will be placed between the FORM tags.

In order for a form to function, it first needs to know how to send the information to the server. There are two methods, GET and POST.

*** METHOD="GET"**

This method will append all of the information from a form on to the end of the URL being requested.

*** METHOD="POST"**

This method will transmit all of the information from a form immediately after the requested URL. This is the preferred method.

In addition to a form needing to know how to send the information, it also needs to know where to send the information to be processed. The ACTION attribute will contain the URL to the form processing script or it may contain an email address.

Example Form:

```
<FORM METHOD=post ACTION="/cgi-bin/example.cgi">
<INPUT type="text" size="10">
 <type="Submit" VALUE="Submit">
</FORM>
```

Example Email Form:

```
<FORM ACTION="mailto:you@yourdomain.com">
Name: <INPUT name="Name" value="" size="10">
Email: <INPUT name="Email" value="" size="10">
<INPUT type="submit" value="Submit">
</FORM>
```

The email form will simply process the information that is placed within your form and send it to your email address. A CGI script is not required.

Notice when the ACTION attribute references an email address, you don't have to include the METHOD attribute.

Form Element Attributes:

Method - Determines which http method will be used to send the form's information to the web server. Action - The URL of the form processing script that resides on the server. This script will process the form's information. Enctype - Determines the method used to encode the form's information. This attribute may be left blank (default) unless you're using a file upload field. Target - Specifies the target frame or window for the response page.

Form Element Properties:

- Text boxes
- Hidden
- Password
- Checkbox
- Radio button
- Submit
- Image submit
- Reset

These properties are specified by using the TYPE attribute within the form's INPUT element.

*** INPUT**

<INPUT TYPE="?">

The INPUT element has the following properties that may be used:

TYPE - Type of input field

NAME - Variable name sent to the form processing script.

VALUE - Information associated with the variable name to be sent to the form processing script.

MAXLENGTH - Maximum number of characters that may be placed within an input area.

SIZE - The size of the input text area.

CHECKED - Default button or box selection.

*** TEXT BOXES**

<INPUT TYPE="text">

Enables users to input text such as an email address.

<FORM METHOD=post ACTION="/cgi-bin/example.cgi">

<INPUT type="TEXT" size="10" maxlength="30">

<INPUT type="Submit" VALUE="Submit">

</FORM>

Text Box Attributes

TYPE - Text

SIZE - The size of the text box specified in characters.

NAME - Name of the variable to be processed by the form processing script.

VALUE - Will display a default value within the text box.

MAXLENGTH - Maximum number of characters that may be placed within the text box.

*** HIDDEN**

<INPUT TYPE="hidden">

Used to send information to the form processing script that you don't want your visitors to see. Nothing will show through the browser.

<INPUT type="hidden" name="redirect"

value="http://www.yourdomain.com/">

Hidden Attributes

TYPE - Hidden

NAME - Name of the variable to be processed by the form processing script.

VALUE - The value of the hidden name expected by the form processing script.

*** PASSWORD**

<INPUT TYPE="password">

Used to enable users to enter a password. When a password is typed in, asterisks will appear instead of text.

<FORM METHOD=post ACTION="/cgi-bin/example.cgi">

<INPUT type="password" size="10" maxlength="30">

<INPUT type="Submit" VALUE="Submit">

</FORM>

Password Attributes

TYPE - Password

NAME - Name of the variable to be processed by the form processing script.

VALUE - Usually blank.

SIZE - The size of the text box specified in characters.

MAXLENGTH - Maximum number of characters that may be placed within the text box.

*** CHECKBOX**

<INPUT TYPE="checkbox">

Enables the user to select multiple options.

<FORM METHOD=post ACTION="/cgi-bin/example.cgi">

<INPUT type="CHECKBOX" name="selection1"> Selection 1 <INPUT type="CHECKBOX" name="selection2"> Selection 2 <INPUT type="CHECKBOX" name="selection3"> Selection 3 <INPUT type="Submit" value="Submit"> </FORM>

Check Box Attributes

TYPE - Checkbox
CHECKED - Specifies a default selection.
NAME - Name of the variable to be processed by the form processing script.
VALUE - The value of the selected check box.
In the next part of this series, we will finish the form element properties and move on to some more advanced form options. Make sure you don't miss this powerful series.

A. ADVANCED HTML FORMS

In part one of this series we focused on setting up a basic form on your website. We discussed the Form Element Attributes and began the Form Element Properties. In part two of this series, we will continue with the Form Element Properties and move on to some more advanced form options.

In part one of this series, we went over Text boxes, Hidden, Password and the Checkbox Form Element Properties. We will now continue with the remaining properties.

*** RADIO BUTTON**

<INPUT type="radio">

Enables the user to select multiple options.

<FORM METHOD=post ACTION="/cgi-bin/example.cgi"> <INPUT type="RADIO" name="selection1"> Selection 1 <INPUT type="RADIO" name="selection2"> Selection 2 <INPUT type="RADIO" name="selection3"> Selection 3 <INPUT type="Submit" value="Submit"> </FORM>

Radio Button Attributes:

TYPE - Radio CHECKED - Specifies a default selection. NAME - Name of the variable to be processed by the form processing script. VALUE - The value of the selected radio button.

*** SUBMIT**

<INPUT type="submit">

Enables users to submit the form information to the form processing script.

<INPUT type="SUBMIT" value="Submit">

Submit Attributes:

TYPE - Submit NAME - Name of the variable to be processed by the form processing script. VALUE - Specifies the text to be displayed on the submit button.

*** IMAGE SUBMIT BUTTON**

<INPUT type="image" SRC="url">

Enables users to submit the form information to the form processing script. Instead of the regular submit button, an image submit button will be displayed.

<INPUT type="image" name="submit" SRC="image.gif">

Image Submit Attributes:

TYPE - Image NAME - Name of the variable to be processed by the form processing script. SRC - Image URL.

*** RESET**

<INPUT type="reset">

Enables users to clear a form if necessary.

<INPUT type="RESET" value="Reset">

Reset Submit Attributes:

TYPE - Reset VALUE - Specifies the text to be displayed on the reset button.

*** SELECT**

<select></select>

Surrounds the code for a selection drop down menu.

<FORM METHOD=post ACTION="/cgi-bin/example.cgi"> <SELECT SIZE="5"> <OPTION>option 1 <OPTION>option 2 <OPTION>option 3 </SELECT> <INPUT type="Submit" value="Submit"> </FORM>

Select Attributes:

NAME - Name of the variable to be processed by the form processing script. SIZE - Specifies the number of visible selections. MULTIPLE - Enables users to select multiple selections.

*** OPTION**

<option>

Used with the SELECT element to display the options.

<FORM METHOD=post ACTION="/cgi-bin/example.cgi"> <SELECT> <OPTION>option 1 <OPTION>option 2 <OPTION>option 3 </SELECT> <INPUT type="Submit" VALUE="Submit"> </FORM>

Option Attributes:

SELECTED - Specifies a default selection. VALUE - Specifies the value of the variable in the select element.

*** TEXTAREA**

<textarea></textarea>

Specifies an open text area.

<FORM METHOD=post ACTION="/cgi-bin/example.cgi"> Enter Your Comments:
 <TEXTAREA wrap="virtual" name="Comments" rows=3 cols=20 maxlength=100> </TEXTAREA>
 <INPUT type="Submit" VALUE="Submit"> <INPUT type="Reset" VALUE="Clear"> </FORM>

Textarea Attributes:

NAME - Name of the variable to be processed by the form processing script. COLS - The number of columns within the text area. ROWS - The number of rows within the text area. WRAP - Specifies the text wrap. The default setting is off. The WRAP can be set to "VIRTUAL" or "PHYSICAL" and will wrap the text as the user types.

Tip: In order to properly format your form, you may want to place it within a table.

Here is a basic email form set up within a table:

<FORM action="mailto:you@yourdomain.com"> <TABLE BORDER="0" CELLPADDING="2"> <TR> <TD><FONT face="Verdana" size=2>Name:</FONT></TD>

<TD><INPUT name="Name" value="" size="10"></TD> </TR> <TR> <TD><FONT face="Verdana" size=2>Email:</FONT></TD> <TD><INPUT name="Email" value="" size="10"></TD> </TR> <TR> <TD></TD> <TD><INPUT type="submit" value="Submit"></TD> </TR> </TABLE> </FORM>

*** Advanced Forms**

If you have a good form processing script, you will have the option to create highly technical forms with additional options:

Multi-page Forms

Provides you with the ability to create a form that spans more than one page. The data you specify will be collected on the first form page and will be transferred to the second page. You can have as many pages as you need and the data will continue to be passed through each page until the final submission. Placeholders are used within each form page to collect and pass the data.

Customized Confirmation Page

Enables you to create a customized confirmation page that may contain your visitor's name and any other information you've collected. In addition, you can even include the date, time and your visitor's IP address (Internet Provider).

Printable Confirmation Page

Enables you to provide your customers with a printable confirmation page for data such as order receipts.

Templates

Provides you with the ability to completely customize the information your form processes. You can use a template to specify how your data will be displayed when it is sent to your email address, and even use a template to set up a database in a specific format.

Database

Enables you to collect your form's data and stores it within a database.

The possibilities are endless. Keep in mind, most form processing scripts will not provide you with these abilities.

The best form I have found is called, Master Form. This form will enable you to have the results emailed to you or to a specified address, can write your information to a database file

and even have a personalized thank you page. In addition, you can even have multi-page forms with no limit on the number of pages. This script costs $35 and can be found here: http://www.web-source.net/cgi-bin/web/jump.cgi?ID=762

You can find a free script with similar features here: http://cgi.tj/scripts/alienform/

In the final part of this series, we will be focusing on some great tips and tricks you can use to spice up your forms such as:

- Creating a Default Form Option
- Customizing Your Input Boxes
- Adding Colour to Your Input Boxes
- Disappearing Form Text
- Flashing Cursor in Form on Load
- Tabbing Through Forms
- Customizing Form Colours

Make sure you don't miss the final lesson in this powerful series.

B. CONTROLS

Users interact with forms through named *controls*. A control's *"control name"* is given by its name attribute. The scope of the name attribute for a control within a FORM element is the FORM element.

Each control has both an initial value and a current value, both of which are character strings. Please consult the definition of each control for information about initial values and possible constraints on values imposed by the control. In general, a control's *"initial value"* may be specified with the control element's value attribute. However, the initial value of a TEXTAREA element is given by its contents, and the initial value of an OBJECT element in a form is determined by the object implementation (i.e., it lies outside the scope of this specification).

The control's *"current value"* is first set to the initial value. Thereafter, the control's current value may be modified through user interaction and scripts.

A control's initial value does not change. Thus, when a form is reset, each control's current value is reset to its initial value. If a control does not have an initial value, the effect of a form reset on that control is undefined.

When a form is submitted for processing, some controls have their name paired with their current value and these pairs are submitted with the form. Those controls for which name/value pairs are submitted are called successful controls.

C. CONTROL TYPES

HTML defines the following control types:

Buttons

Authors may create three types of buttons:

- Submit buttons: When activated, a submit button submits a form. A form may contain more than one submit button.
- Reset buttons: When activated, a reset button resets all controls to their initial values.
- Push buttons: Push buttons have no default behaviour. Each push button may have client-side scripts associated with the element's event attributes. When an event occurs (e.g., the user presses the button, releases it, etc.), the associated script is triggered.

  Authors should specify the scripting language of a push button script through a default script declaration (with the META element).

Authors create buttons with the BUTTON element or the INPUT element. Please consult the definitions of these elements for details about specifying different button types.

Note. *Authors should note that the BUTTON element offers richer rendering capabilities than the INPUT element.*

Checkboxes

Checkboxes (and radio buttons) are on/off switches that may be toggled by the user. A switch is "on" when the control element's checked attribute is set. When a form is submitted, only "on" checkbox controls can become successful.

Several checkboxes in a form may share the same control name. Thus, for example, checkboxes allow users to select several values for the same property. The INPUT element is used to create a checkbox control.

Radio buttons

Radio buttons are like checkboxes except that when several share the same control name, they are mutually exclusive: when one is switched "on", all others with the same name are switched "off". The INPUT element is used to create a radio button control.

If no radio button in a set sharing the same control name is initially "on", user agent behaviour for choosing which control is initially "on" is undefined. **Note.** Since existing implementations handle this case differently, the current specification differs from RFC 1866 ([RFC1866] section 8.1.2.4), which states:
At all times, exactly one of the radio buttons in a set is checked. If none of the <INPUT> elements of a set of radio buttons specifies `CHECKED', then the user agent must check the first radio button of the set initially.

Since user agent behavior differs, authors should ensure that in each set of radio buttons that one is initially "on".

Menus

Menus offer users options from which to choose. The SELECT element creates a menu, in combination with the OPTGROUP and OPTION elements.

Text input

Authors may create two types of controls that allow users to input text. The INPUT element creates a single-line input control and the TEXTAREA element creates a multi-line input control. In both cases, the input text becomes the control's current value.

File select

This control type allows the user to select files so that their contents may be submitted with a form. The INPUT element is used to create a file select control.

Hidden controls

Authors may create controls that are not rendered but whose values are submitted with a form. Authors generally use this control type to store information between client/server exchanges that would otherwise be lost due to the stateless nature of HTTP (see [RFC2616]). The INPUT element is used to create a hidden control.

Object controls

Authors may insert generic objects in forms such that associated values are submitted along with other controls. Authors create object controls with the OBJECT element.

The elements used to create controls generally appear inside a FORM element, but may also appear outside of a FORM element declaration when they are used to build user interfaces. This is discussed in the section on intrinsic events. Note that controls outside a form cannot be successful controls.

SCRIPTS

A client-side *script* is a program that may accompany an HTML document or be embedded directly in it. The program executes on the client's machine when the document loads, or at some other time such as when a link is activated. HTML's support for scripts is independent of the scripting language.

Scripts offer authors a means to extend HTML documents in highly active and interactive ways. For example:

- Scripts may be evaluated as a document loads to modify the contents of the document dynamically.
- Scripts may accompany a form to process input as it is entered. Designers may dynamically fill out parts of a form based on the values of other fields. They may also ensure that input data conforms to predetermined ranges of values, that fields are mutually consistent, etc.
- Scripts may be triggered by events that affect the document, such as loading, unloading, element focus, mouse movement, etc.
- Scripts may be linked to form controls (e.g., buttons) to produce graphical user interface elements.

There are two types of scripts authors may attach to an HTML document:

- Those that are executed one time when the document is loaded by the user agent. Scripts that appear within a SCRIPT element are executed when the document is loaded. For user agents that cannot or will not handle scripts, authors may include alternate content via the NOSCRIPT element.
- Those that are executed every time a specific event occurs. These scripts may be assigned to a number of elements via the intrinsic event attributes.

As HTML does not rely on a specific scripting language, document authors must explicitly tell user agents the language of each script. This may be done either through a default declaration or a local declaration.

The default scripting language

Authors should specify the default scripting language for all scripts in a document by including the following META declaration in the HEAD:

```
<META http-equiv="Content-Script-Type" content="type">
```

where "type" is a content type naming the scripting language. Examples of values include "text/tcl", "text/javascript", "text/vbscript".

In the absence of a META declaration, the default can be set by a "Content-Script-Type" HTTP header.

```
Content-Script-Type: type
```

where "type" is again a content type naming the scripting language.

User agents should determine the default scripting language for a document according to the following steps (highest to lowest priority):

1. If any META declarations specify the "Content-Script-Type", the last one in the character stream determines the default scripting language.
2. Otherwise, if any HTTP headers specify the "Content-Script-Type", the last one in the character stream determines the default scripting language.

Documents that do not specify default scripting language information and that contain elements that specify an intrinsic event script are incorrect. User agents may still attempt to interpret incorrectly specified scripts but are not required to. Authoring tools should generate default scripting language information to help authors avoid creating incorrect documents.

Local declaration of a scripting language

The type attribute must be specified for each SCRIPT element instance in a document. The value of the type attribute for a SCRIPT element overrides the default scripting language for that element.

In this example, we declare the default scripting language to be "text/tcl". We include one SCRIPT in the header, whose script is located in an external file and is in the scripting language "text/vbscript". We also include one SCRIPT in the body, which contains its own script written in "text/javascript".

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
     "http://www.w3.org/TR/html4/strict.dtd">
<HTML>
<HEAD>
<TITLE>A document with SCRIPT</TITLE>
<META http-equiv="Content-Script-Type" content="text/tcl">
<SCRIPT type="text/vbscript" src="http://someplace.com/progs/vbcalc">
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT type="text/javascript">
```

```
...some JavaScript...
</SCRIPT>
</BODY>
</HTML>
```

References to HTML elements from a script

Each scripting language has its own conventions for referring to HTML objects from within a script. This specification does not define a standard mechanism for referring to HTML objects.

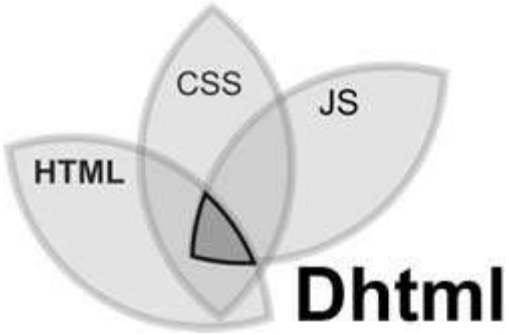
However, scripts should refer to an element according to its assigned name. Scripting engines should observe the following precedence rules when identifying an element: a nameattribute takes precedence over an id if both are set. Otherwise, one or the other may be used.

1.4 DYNAMIC HTML AND XML

In this section we shall cover Dynamic HTML and XML.

1.4.1 DHTML

DHTML is essentially Dynamic HTML. It is a new way of looking at and controlling the standard HTML codes and commands. DHTML is a collection of technologies that are used to create interactive and animated web sites. XML stands for Extensible Markup Language. It is a specification developed by the W3C. It is a markup language designed especially for Web documents. It defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.



DHTML is essentially Dynamic HTML. It is a new way of looking at and controlling the standard HTML codes and commands. DHTML is a collection of technologies that are used to create interactive and animated web sites. DHTML gives more control over the HTML elements. It allows one to incorporate a client-side scripting language, such as JavaScript, a presentation definition language, such as CSS, and the Document Object Model in HTML web pages.

DHTML also allows the pages to change at any time, without returning to the Web server first. It allows scripting languages to change a web page's look and function after the page has been fully loaded and during the viewing process. It also allows the user to add effects to their pages that are otherwise difficult to achieve.

Wikipedia list additional DHTML features, such as DHTML allows the developers to:

- Animate text and images in their document, independently moving each element from any starting point to any ending point, following a predetermined path or one chosen by the user.
- Embed a ticker that automatically refreshes its content with the latest news, stock quotes, or other data.

- Use a form to capture user input, and then process, verify and respond to that data without having to send data back to the server.
- Include rollover buttons or drop-down menus.

The Concepts and Features in Dynamic HTML

- An object-oriented view of a Web page and its elements
- Cascading style sheets and the layering of content
- Programming that can address all or most page elements
- Dynamic fonts

1. An Object-Oriented View of Page Elements

Each page element (division or section, heading, paragraph, image, list, and so forth) is viewed as an "object." (Microsoft calls this the "Dynamic HTML Object Model." Netscape calls it the "HTML Object Model." W3C calls it the "Document Object Model.") For example, each heading on a page can be named, given attributes of text style and color, and addressed by name in a small program or "script" included on the page. This heading or any other element on the page can be changed as the result of a specified event such a mouse passing over or being clicked or a time elapsing. Or an image can be moved from one place to another by "dragging and dropping" the image object with the mouse. (These event possibilities can be viewed as the reaction capabilities of the element or object.) Any change takes place immediately (since all variations of all elements or objects have been sent as part of the same page from the Web server that sent the page). Thus, variations can be thought of as different properties of the object.

Not only can element variations change text wording or color, but everything contained within a heading object can be replaced with new content that includes different or additional HTML as well as different text. Microsoft calls this the "Text Range technology."

Although JavaScript, Java applet, and ActiveX controls were present in previous levels of Web pages, dynamic HTML implies an increased amount of programming in Web pages since more elements of a page can be addressed by a program.

A feature called dynamic fonts lets Web page designers include font files containing specific font styles, sizes, and colours as part of a Web page and to have the fonts downloaded with the page. That is, the font choice no longer is dependent on what the user's browser provides.

2. Style Sheets and Layering

A describes the default style characteristics (including the page layout and font type style and size for text elements such as headings and body text) of a document or a portion of a document. For Web pages, a style sheet also describes the default background color or image, hypertext link colours, and possibly the content of page. Style sheets help ensure consistency across all or a group of pages in a document or a Web site.

Dynamic HTML includes the capability to specify style sheets in a "cascading style sheet" fashion (that is, linking to or specifying different style sheets or style statements with predefined levels of precedence within the same or a set of related pages). As the result of user interaction, a new style sheet can be made applicable and result in a change of appearance of the Web page. You can have multiple layers of style sheet within a page, a style sheet within a style sheet within a style sheet. A new style sheet may only vary one element from the style sheet above it.

Layering is the use of alternate style sheets or other approaches to vary the content of a page by providing content layers that can overlay (and replace or superimpose on) existing content sections. Layers can be programmed to appear as part of a timed presentation or as the result of user interaction.

HOW DHTML WORKS

DHTML is a combination of HTML, Cascading Style Sheets, JavaScript, and the Document Object Model. Example below illustrates how these elements work together. The web page shown here uses simple DHTML to change the style of links to be red and underlined when the mouse is rolled over them. You can use this basic format to tie CSS styles to common events like `onMouseOver` or `OnClick`, so you can change the styles of most elements on the fly.

Example. Rollover style changes using DHTML

```
<html>    (A)
<head>
<title>Rollover Style Changes</title>

<style>    (B)
<!--
a { text-decoration: none; }
-->
</style>

<script>    (C)
<!--
function turnOn(currentLink) {
    currentLink.style.color = "#990000";    (D)
    currentLink.style.textDecoration = "underline";
}
```

```
function turnOff(currentLink) {
    currentLink.style.color = "#0000FF";
    currentLink.style.textDecoration = "none";
}
//-->
</script>
</head>

<body bgcolor="#FFFFFF">
<a href="#home"   (E)
   onMouseOver="turnOn(this);" onMouseOut="turnOff(this);">Home</a>
<a href="#contact"
   onMouseOver="turnOn(this);" onMouseOut="turnOff(this);">Contact</a>
<a href="#links"
   onMouseOver="turnOn(this);" onMouseOut="turnOff(this);">Links</a>
</body>
</html>
```

This page is an HTML file, so it starts with normal <html> and <head> HTML tags.

A. In the <head>, we have a CSS style sheet, defined using the <style> tag, that removes any text decorations from all the links in the document. In this case, the point is to remove the default underlines from links.
B. Inside the <script> tag, there are two JavaScript functions, turnOn( ) and turnOff( ), that change the style of a link when the user moves the mouse over and back out of the link. When the mouse enter a link, the text is underlined and turned red. When the mouse exits, these effects are removed.
C. The script uses the DOM to reference the link's style attribute and change the color and textDecoration properties, which are the DOM equivalents of the CSS properties color and text-decoration.
D. In this <a> tag, the onMouseOver and onMouseOut event handlers are used to set up the calls to turnOn( ) and turnOff( ).

1.4.2 XML

XML stands for Extensible Mark-up Language. It is a specification developed by the W3C. It is a mark-up language designed especially for Web documents. It defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. It allows designers to create their own customized tags. It also enables the definition, transmission, validation, and interpretation of data between applications and organizations.

XML is a text-based data format with strong support via Unicode for languages. It emphasizes simplicity, generality, and usability over the Internet. It is also widely used for the representation of arbitrary data structures, especially in web services. Programmers often use APIs while processing XML data and schema systems to aid in the definition of XML-based languages.

The XML syntax has formed the basis for many document formats, such as RSS, Atom, SOAP, and XHTML. In fact, XML-based formats have become the default for many office-productivity tools, including Microsoft Office, OpenOffice.org and LibreOffice, and Apple's iWork.

Some differences between DHTML and XML:

- DHTML is used to position information in a web page, and XML is used to describe that information.
- DHTML is HTML with JavaScript actions, whereas XML is more of universal way to transport info than a mark-up language.
- DHTML is used to display the dynamic web site pages, whereas XML is a mark-up language designed especially for Web documents.
- XML is an extensible mark-up language that was developed to retain the flexibility and power of HTML while reducing most of the complexity.

Concepts of XML

The building blocks of XML documents are Elements, Tags, and Attributes.

- **Elements**

  Elements are the main building blocks of XML documents.

  Examples of XML elements in pmiod.xsd are component or code. Elements can contain text, other elements, or be empty.

  The values of minOccurs and maxOccurs for an element in the Schema file indicate how many times this element must occur in the corresponding XML file; if minOccurs and maxOccurs are not specified, the element must appear once.

- **Tags**

  Tags are used to markup elements.

  In the XML file, a starting tag like <element_name> marks up the beginning of an element, and an ending tag like </element_name> marks up the end of an element.

  Example: <laboratory>Meteo-France</laboratory>

  An empty element will appear as <element_name />.

- **Attributes**

Attributes provide extra information about elements and are placed inside the start tag of an element. As indicated in the Schema file, an attribute may be ``required'' (use='required') or ``optional'' (use='optional').

Example: <grid local_name="AT31_2D" >

The name of the element is ``grid''. The name of the attribute is ``local_name''. The value of the attribute is ``AT31_2D''.

SESSION 2: CONNECTING WEBSITES TO BUSINESS APPLICATIONS

On completion of this section you will be able to demonstrate an understanding of connecting web sites to business applications

1. The demonstration explains the usage of CGI
2. The demonstration describes the types of services provided by database backed web sites
3. The demonstration discusses the underlying technologies that link databases and web sites
4. The demonstration describes known methods of linking web pages to back-end proprietary applications

2.1 USAGE OF CGI

For us to understand CGI lets us first look at how the web works. When you browse the web the situation is basically this: you sit at your computer and want to see a document somewhere on the web, to which you have the URL.

Since the document you want to read is somewhere else in the world and probably very far away from you some more details are needed to make it available to you. The first detail is your browser. You start it up and type the URL into it (at least you tell the browser somehow where you want to go, perhaps by clicking on a link).

However, the picture is still not complete, as the browser can't read the document directly from the disk where it's stored if that disk is on another continent. So for you to be able to read the document the computer that contains the document must run a web server. A web server is a just a computer program that listens for requests from browsers and then execute them.

So what happens next is that the browser contacts the server and requests that the server deliver the document to it. The server then gives a response which contains the document and the browser happily displays this to the user. The server also tells the browser what kind of document this is (HTML file, PDF file, ZIP file etc) and the browser then shows the document with the program it was configured to use for this kind of document.

The browser will display HTML documents directly, and if there are references to images, Java applets, sound clips etc in it and the browser has been set up to display these it will request these also from the servers on which they reside. (Usually the same server as the document, but not always.) It's worth noting that these will be separate requests, and add additional load to the server and network. When the user follows another link the whole sequence starts anew.

These requests and responses are issued in a special language called HTTP, which is short for HyperText Transfer Protocol. What this article basically does is describe how this works. Other common protocols that work in similar ways are FTP and Gopher, but there are also protocols that work in completely different ways. None of these are covered here, sorry.
It's worth noting that HTTP only defines what the browser and web server say to each other, not how they communicate. The actual work of moving bits and bytes back and forth across the network is done by TCP and IP, which are also used by FTP and Gopher (as well as most other internet protocols).

When you continue, note that any software program that does the same as a web browser (ie: retrieve documents from servers) is called a client in network terminology and a user agent in web terminology. Also note that the server is properly the server program, and not the computer on which the server is an application program. (Sometimes called the server machine)

What happens when I follow a link?

Step 1: Parsing the URL

The first thing the browser has to do is to look at the URL of the new document to find out how to get hold of the new document. Most URLs have this basic form: "protocol://server/request-URI". The protocol part describes how to tell the server which document the you want and how to retrieve it. The server part tells the browser which server to contact, and the request-URI is the name used by the web server to identify the document. (I use the term request-URI since it's the one used by the HTTP standard, and I can't think of anything else that is general enough to not be misleading.)

Step 2: Sending the request

Usually, the protocol is "http". To retrieve a document via HTTP the browser transmits the following request to the server: "GET /request-URI HTTP/version", where version tells the server which HTTP version is used. (Usually, the browser includes some more information as well. The details are covered **later**.)

One important point here is that this request string is all the server ever sees. So the server doesn't care if the request came from a browser, a link checker, a validator, a search engine robot or if you typed it in manually. It just performs the request and returns the result.

Step 3: The server response

When the server receives the HTTP request it locates the appropriate document and returns it. However, an HTTP response is required to have a particular form. It must look like this:

```
HTTP/[VER] [CODE] [TEXT]
Field1: Value1
Field2: Value2

...Document content here...
```

The first line shows the HTTP version used, followed by a three-digit number (the HTTP status code) and a reason phrase meant for humans. Usually the code is 200 (which basically means that all is well) and the phrase "OK". The first line is followed by some lines called the header, which contains information about the document. The header ends with a blank line, followed by the document content. This is a typical header:

```
HTTP/1.0 200 OK
Server: Netscape-Communications/1.1
```

```
Date: Tuesday, 25-Nov-97 01:22:04 GMT
Last-modified: Thursday, 20-Nov-97 10:44:53 GMT
Content-length: 6372
Content-type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML>
...followed by document content...
```

We see from the first line that the request was successful. The second line is optional and tells us that the server runs the Netscape Communications web server, version 1.1. We then get what the server thinks is the current date and when the document was modified last, followed by the size of the document in bytes and the most important field: "Content-type".

The content-type field is used by the browser to tell which format the document it receives is in. HTML is identified with "text/html", ordinary text with "text/plain", a GIF is "image/gif" and so on. The advantage of this is that the URL can have any ending and the browser will still get it right.

An important concept here is that to the browser, the server works as a black box. Ie: the browser requests a specific document and the document is either returned or an error message is returned. How the server produces the document remains unknown to the browser. This means that the server can read it from a file, run a program that generates it, compile it by parsing some kind of command file or (very unlikely, but in principle possible) have it dictated by the server administrator via speech recognition software. This gives the server administrator great freedom to experiment with different kinds of services as the users don't care (or even know) how pages are produced.

What the server does

When the server is set up it is usually configured to use a directory somewhere on disk as its root directory and that there be a default file name (say "index.html") for each directory. This means that if you ask the server for the file "/" (as in "http://www.domain.tld/") you'll get the file index.html in the server root directory. Usually, asking for "/foo/bar.html" will give you the bar.html file from the foo directory directly beneath the server root.

Usually, that is. The server can be set up to map "/foo/" into some other directory elsewhere on disk or even to use server-side programs to answer all requests that ask for that directory. The server does not even have to map requests onto a directory structure at all, but can use some other scheme.

HTTP versions

So far there are three versions of HTTP. The first one was HTTP/0.9, which was truly primitive and never really specified in any standard. This was corrected by HTTP/1.0, which was issued as a standard in RFC 1945. HTTP/1.0 is the version of HTTP that is in common use today (usually with some 1.1 extensions), while HTTP/0.9 is rarely, if ever, used by browsers. (Some simpler HTTP clients still use it since they don't need the later extensions.)

RFC 2068 describes HTTP/1.1, which extends and improves HTTP/1.0 in a number of areas. Very few browsers support it (MSIE 4.0 is the only one known to the author), but servers are beginning to do so.

The major differences are a some extensions in HTTP/1.1 for authoring documents online via HTTP and a feature that lets clients request that the connection be kept open after a request so that it does not have to be reestablished for the next request. This can save some waiting and server load if several requests have to be issued quickly.

This document describes HTTP/1.0, except some sections that cover the HTTP/1.1 extensions. Those will be explicitly labeled.

The request sent by the client

The shape of a request

Basically, all requests look like this:

```
[METH] [REQUEST-URI] HTTP/[VER]
[fieldname1]: [field-value1]
[fieldname2]: [field-value2]

[request body, if any]
```

The METH (for request method) gives the request method used, of which there are several, and which all do different things. The above example used GET, but below some more are explained. The REQUEST-URI is the identifier of the document on the server, such as "/index.html" or whatever. VER is the HTTP version, like in the response. The header fields are also the same as in the server response.

The request body is only used for requests that transfer data to the server, such as POST and PUT. (Described below)

Getting a document

There are several request types, with the most common one being GET. A GET request basically means "send me this document" and looks like this: "GET document_path HTTP/version". (Like it was described above.) For the URL **"http://www.yahoo.com/"** the

document_path would be "/", and for **"http://www.w3.org/Talks/General.html"** it is "/Talks/General.html".

However, this first line is not the only thing a user agent (*UA*) usually sends, although it's the only thing that's really necessary. The UA can include a number of *header fields* in the request to give the server more information. These fields have the form "fieldname: value" and are all put on separate lines after the first request line.

Some of the header fields that can be used with GET are:

User-Agent

This is a string identifying the user agent. An English version of Netscape 4.03 running under Windows NT would send "Mozilla/4.03 [en] (WinNT; I ;Nav)". (Mozilla is the old name for Netscape. See the **references** for more details.)

Referer

The referer field (yes, it's misspelled in the standard) tells the server where the user came from, which is very useful for logging and keeping track of who links to ones pages.

If-Modified-Since

If a browser already has a version of the document in its **cache** it can include this field and set it to the time it retrieved that version. The server can then check if the document has been modified since the browser last downloaded it and send it again if necessary. The whole point is of course that if the document *hasn't* changed, then the server can just say so and save some waiting and network traffic.

From

This header field is a spammers dream come true: it is supposed to contain the email address of whoever controls the user agent. Very few, if any, browsers use it, partly because of the threat from spammers. However, web robots should use it, so that webmasters can contact the people responsible for the robot should it misbehave.

Authorization

This field can hold username and password if the document in question requires **authorization** to be accessed.

To put all these pieces together: this is a typical GET request, as issued by my browser (Opera):

```
GET / HTTP/1.0
User-Agent: Mozilla/3.0 (compatible; Opera/3.0; Windows 95/NT4)
Accept: */*
Host: birk105.studby.uio.no:81
```

HEAD: checking documents

One may sometimes want to see the headers returned by the server for a particular document, without actually downloading the document. This is exactly what the HEAD request method provides. HEAD looks and works exactly like GET, only with the difference that the server only returns the headers and not the document content.

This is very useful for programs like link checkers, people who want to see the response headers (to see what server is used or to verify that they are correct) and many other kinds of uses.

Playing web browser

You can actually play web browser yourself and write HTTP requests directly to web servers. This can be done by telnetting to port 80, writing the request and hitting enter twice, like this:

```
larsga - tyrfing>telnet www.w3.org 80
Trying 18.23.0.23...
Connected to www.w3.org.
Escape character is '^]'.
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Tue, 17 Feb 1998 22:24:53 GMT
Server: Apache/1.2.5
Last-Modified: Wed, 11 Feb 1998 18:22:22 GMT
ETag: "2c3136-23c1-34e1ec5e"
Content-Length: 9153
Accept-Ranges: bytes
Connection: close
Content-Type: text/html; charset=ISO-8859-1

Connection closed by foreign host.
larsga - tyrfing>
```

However, this works best under Unix as the Windows telnet clients I've used are not very suitable for this (and hard to set up so that it works).

The response returned by the server

Outline

What the server returns consists of a line with the status code, a list of header fields, a blank line and then the requested document, if it is returned at all. Sort of like this:

```
HTTP/1.0 code text
Field1: Value1
Field2: Value2

...Document content here...
```

The status codes

The status codes are all three-digit numbers that are grouped by the first digit into 5 groups. The reason phrases given with the status codes below are just suggestions. Server can return any reason phrase they wish.

1xx: Informational

No 1xx status codes are defined, and they are reserved for experimental purposes only.

2xx: Successful

Means that the request was processed successfully.

200 OK

Means that the server did whatever the client wanted it to, and all is well.

Others

The rest of the 2xx status codes are mainly meant for script processing and are not often used.

3xx: Redirection

Means that the resource is somewhere else and that the client should try again at a new address.

301 Moved permanently

The resource the client requested is somewhere else, and the client should go there to get it. Any links or other references to this resource should be updated.

302 Moved temporarily

This means the same as the 301 response, but links should now not be updated, since the resource may be moved again in the future.

304 Not modified

This response can be returned if the client used the if-modified-since header field and the resource has not been modified since the given time. Simply means that the cached version should be displayed for the user.

4xx: Client error

Means that the client screwed up somehow, usually by asking for something it should not have asked for.

400: Bad request

The request sent by the client didn't have the correct syntax.

401: Unauthorized

Means that the client is not allowed to access the resource. This may change if the client retries with an **authorization** header.

403: Forbidden

The client is not allowed to access the resource and authorization will not help.

404: Not found

Seen this one before? :) It means that the server has not heard of the resource and has no further clues as to what the client should do about it. In other words: dead link.

5xx: Server error

This means that the server screwed up or that it couldn't do as the client requested.

500: Internal server error

Something went wrong inside the server.

501: Not implemented

The request method is not supported by the server.

503: Service unavailable

This sometimes happens if the server is too heavily loaded and cannot service the request. Usually, the solution is for the client to wait a while and try again.

The response header fields

These are the header fields a server can return in response to a request.

Location

This tells the user agent where the resource it requested can be found. The value is just the URL of the new resource.

Server

This tells the user agent which web server is used. Nearly all web servers return this header, although some leave it out.

Content-length

This gives the size of the resource, in bytes.

Content-type

This describes the file format of the resource.

Content-encoding

This means that the resource has been coded in some way and must be decoded before use.

Expires

This field can be set for data that are updated at a known time (for instance if they are generated by a script). It is used to prevent browsers from caching the resource beyond the given date.

Last-modified

This tells the browser when the resource was last modified. Can be useful for mirroring, update notification etc.

Caching: agents between the server and client

The browser cache

You may have noticed that when you go back to a page you've looked at not too long before the page loads much quicker. That's because the browser stored a local copy of it when it was first downloaded. These local copies are kept in what's called a cache. Usually one sets a maximum size for the cache and a maximum caching time for documents.

This means that when a new page is visited it is stored in the cache, and if the cache is full (near the maximum size limit) some document that the browser considers unlikely to be visited again soon is deleted to make room. Also, if you go to a page that is stored in the cache the browser may find that you've set 7 days as a the maximum storage time and 8 days have now passed since the last visit, so the page needs to be reloaded.

Exactly how caches work differ between browsers, but this is the basic idea, and it's a good one because it saves both time for the user and network traffic. There are also some HTTP details involved, but they will be covered later.

Proxy caches

Browser caches are a nice feature, but when many users browse from the same site one usually ends up storing the same document in many different caches and refreshing it over and over for different uses. Clearly, this isn't optimal.

The solution is to let the users share a cache, and this is exactly what proxy caches are all about. Browsers still have their local caches, but HTTP requests for documents not in the browser cache are not sent to the server any more, instead they are sent to the proxy cache. If the proxy has the document in its cache it will just return the document (like the browser cache would), and if it doesn't it will submit the request on behalf of the browser, store the result and relay it to the browser.

So the proxy is really a common cache for a number of users and can reduce network traffic rather dramatically. It can also skew log-based statistics badly. :)

A more advanced solution than a single proxy cache is a hierarchy of proxy caches. Imagine a large ISP may have one proxy cache for each part of the country and set up each of the regional proxies to use a national proxy cache instead of going directly to the source web servers. This solution can reduce network traffic even further.

Server-side programming

Server-side scripts or programs are simply programs that are run on the web server in response to requests from the client. These scripts produce normal HTML (and sometimes HTTP headers as well) as output which is then fed back to the client as if the client had requested an ordinary page. In fact, there is no way for the client software to tell whether scripting has been used or not.

Technologies such as JavaScript, VBScript and Java applets all run in the client and so are not examples of this. There is a major difference between server-side and client-side scripting as the client and server are usually different computers. So if all the data the program needs are located on the server it may make sense to use server-side scripting instead of client-side. (There is also the problem that the client may not have a browser that supports the scripting technology or it may be turned off.) If the program and user need to interact often client-side scripting is probably best, to reduce the number of requests sent to the server.

So: in general, if the program needs a lot of data and infrequent interactions with the server server-side scripting is probably best. Applications that use less data and more interaction are best put in the client. Also, applications that gather data over time need to be on the server where the data file is kept.

An example of the first would be search engines like Altavista. Obviously it's not feasible to download all the documents Altavista has collected to search in them locally. An example of the last would be a simple board game. No data are needed and having to send a new request to the server for each move you make quickly gets tedious.

There is one kind of use that has been left out here: what do you do when you want a program to work on data with a lot of interaction with the user? There is no good solution right now, but there is one on the horizon called XML. (See the **the references** for more info.)

How it works

The details of how server-side scripting works vary widely with the technique used (and there are loads of them). However, some things remain the same. The web server receives a request

just like any other, but notes that this URL does not map to a flat file, but instead somehow to a scripting area.

The server then starts the script, feeding it all the information contained in the request headers and URL. The script then runs and produces as its output the HTML and HTTP headers to be returned to the client, which the server takes care of.

CGI

CGI (Common Gateway Interface) is a way for web servers and server-side programs to interact. CGI is completely independent of programming language, operating system and web server. Currently it is the most common server-side programming technique and it's also supported by almost every web server in existence. Moreover, all servers implement it in (nearly) the same way, so that you can make a CGI script for one server and then distribute it to be run on any web server.

The server needs a way to know which URLs map to scripts and which URLs just map to ordinary HTML files. For CGI this is usually done by creating CGI directories on the server. This is done in the server setup and tells the server that all files in a particular top-level directory are CGI scripts (located somewhere on the disk) to be executed when requested. (The default directory is usually /cgi-bin/, so one can tell that URLs like this: http://www.varsity.edu/cgi-bin/search point to a CGI script. Note that the directory can be called anything.) Some servers can also be set up to not use CGI directories and instead require that all CGI programs have file names ending in .cgi.

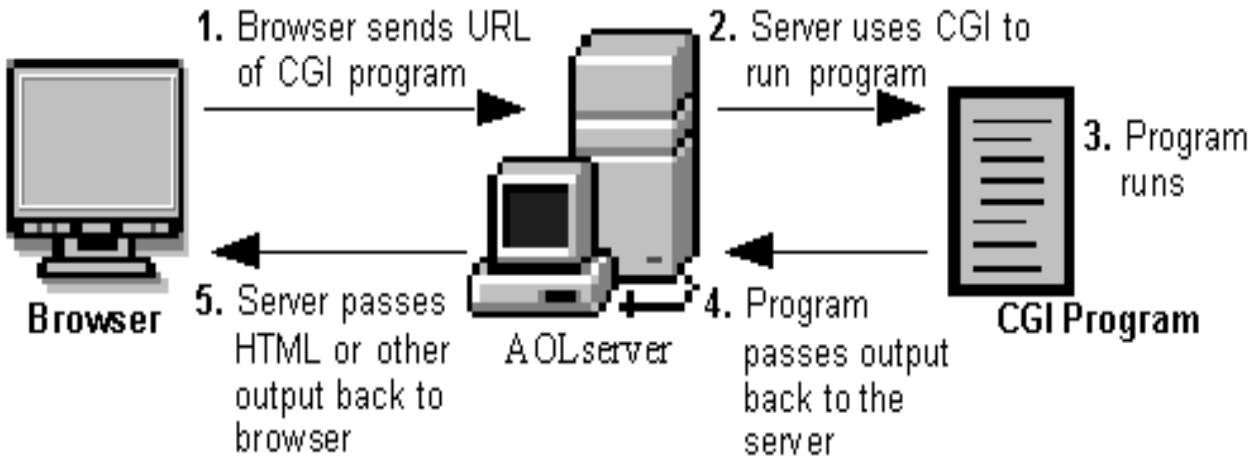
CGI programs are just ordinary executable programs (or interpreted programs written in, say, Perl or Python, as long as the server knows how to start the program), so you can use just about any programming language you want. Before the CGI program is started the web server sets a number of environment variables that contain the information the web server received in the request. Examples of this are the IP address of the client, the headers of the request etc. Also, if the URL requested contained a ?, everything after the ? is put in an environment variable by itself.

This means that extra information about the request can be put into the URL in the link. One way this is often used is by multi-user hit counters to tell which user was hit this time. Thus, the user can insert an image on his/her page and have the SRC attribute be a link to the CGI script like this: SRC="http://stats.vendor.com/cgi-bin/counter.pl?username". Then the script can tell which user was hit and increment and display the correct count. (Ie: that of Peter and not Paul.)

The way the CGI returns its output (HTTP headers and HTML document) to the server is exceedingly simple: it writes it to standard out. In other words, in a Perl or Python script you

just use the print statement. In C you use printf or some equivalent (C++ uses cout <<) while Java would use System.out.println.

Here is a diagram of how a CGI program runs:



For example, suppose you have a form that lets people comment on your Web pages. You want the comments emailed to you and you want to automatically generate a page and send it back to your reader.

1. The reader fills out your form and clicks the "Submit" button. The <FORM> tag in your page might look like this:

   ```
   <FORM METHOD="POST" ACTION="/cgi-bin/myprog">
   ```

   The METHOD controls how the information typed into the form is passed to your program. It can be "GET" or "POST"

   The ACTION determines which program should be run.

   Other ways for a reader to run a program are by providing a direct link to the program without allowing the reader to supply any variables through a form, or by using the <ISINDEX> tag.

2. When AOLserver gets a request for a URL that maps to a CGI directory or a CGI file extension (as defined in the configuration file,it starts a separate process and runs the program within that process. The AOLserver also sets up a number of environment variable within that process. These environment variables include some standard CGI variables, and optionally any variables you define in the configuration file for this type of program.
3. The program runs. The program can be any type of executable program. For example, you can use C, C++, Perl, Unix shell scripts, or Fortran.

   In this example, the program takes the comments from the form as input and sends them to you as email. If the form method is "GET", it gets the input from an environment

variable. If the form method is "POST", it gets the input from standard input. It also assembles a HTML page and sends it to standard output.

4. Any information the program passes to standard output is automatically sent to the AOLserver when the program finishes running.
5. The server adds any header information needed to identify the output and sends it back to the reader's browser, which displays the output.

A CGI program would not do the following.

1. it does not interact with a user directly.
2. It does not interact directly with a web browser or a graphical interface. In other words it does not display or retrieve information from menus, commands and other interactive features of a client browser.
3. It does not create graphics or windows by itself.

A program must meet the following criteria to be qualified as a CGI program.

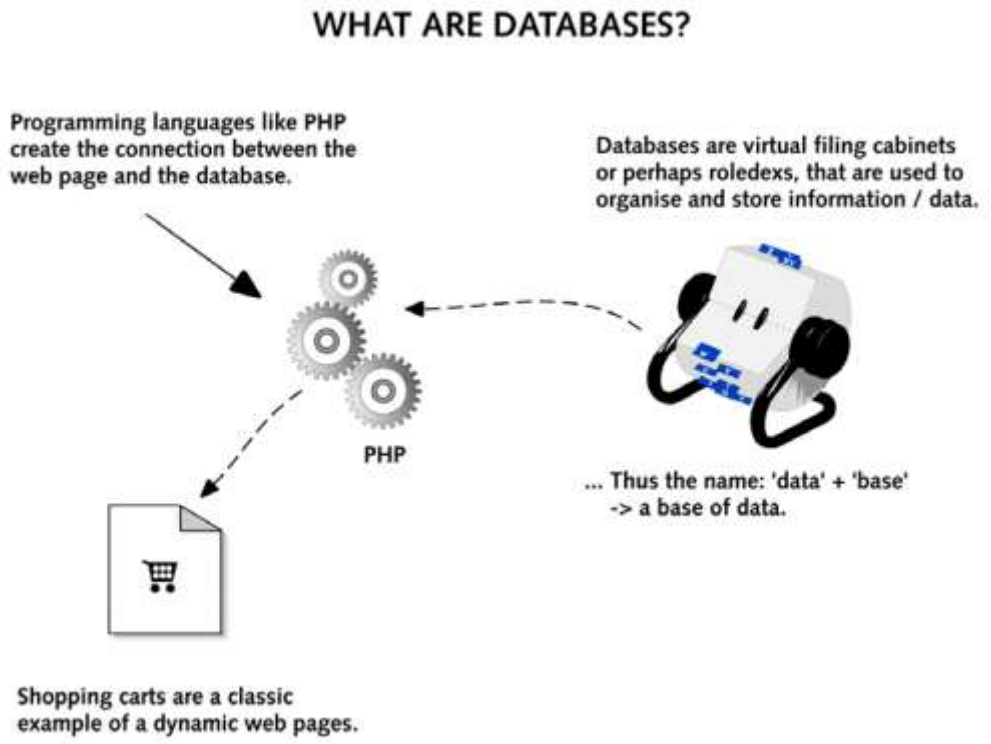
1. One should be able to execute it directly by typing its name from the command line. A java program does not qualify to be a CGI program because it cannot be executed in the java virtual machine unless we type "java program name" in the command prompt.
2. The program should generate a valid content type header.1 Following are some of the advantages and disadvantages of CGI programs.

Advantages

- CGI programs are portable and work on a wide variety of web servers and platforms.
- They are language independent. You can write them in any language and make them work in a wide variety of environments. Some of the programming languages good enough to write your CGI scripts are Perl, UNIX Shell, C language, Visual Basic, Python, C# and Java.
- They provide simple interfaces for the clients to interact with the web servers.
- They are scalable programs and you can use them to perform simple tasks in the application layer as well as more complex tasks such as interacting with databases and shopping carts. 5. They provide interactivity to a web application and enhance user experience

2.2 TYPES OF SERVICES PROVIDED BY DATABASE BACKED WEBSITES

One of the most common types of dynamic web pages is the database driven type. This means that you have a web page that grabs information from a database (the web page is connected to the database by programming,) and inserts that information into the web page each time it is loaded. If the information stored in the database changes, the web page connected to the database will also change accordingly (and automatically,) without human intervention. This is commonly seen on online banking sites where you can log in (by entering your user name and password,) and check out your bank account balance. Your bank account information is stored in a database and has been connected to the web page with programming (ex: PHP, ASP) thus enabling you to see your banking information.



Imagine if the web page holding your banking information had to be built traditionally (that is by hand,) every time your bank balance changed! Even a thousand monkeys working 24/7 drinking 5 cups of coffee a day, would not be able to keep up!

Common database driven websites include:

- **E-Commerce platforms:** These businesses leverage data driven websites because of the expected changes in prices, offers and services. This guarantees the information internet users find is always fresh and up-to-date.
- **Content Management Systems (CMS):** If the website is going to use a CMS then it is database driven. Users can easily update content on the website even without the need for any specialized programming skills. These CMSs include WordPress and Joomla and they have an easy-to-use editor to allow publishing of content, editing and deleting.
- **Blogs:** Most blogs and online community forums are database driven because they involve regular updates by users. Whether people are leaving comments or liking a website there is immediate change on the page.

Advantages of a Database-Driven Website

- It is easier and faster to update content. A few clicks are all the webmaster needs to update the content. Changes are made almost in real-time
- It is ideal for an ecommerce site where different products need to be added, prices changes and offers introduced.
- No need for specialized HTML knowledge or expertise to change content on the website.
- High scalability: Every business grows with time and a database driven website offers room for growth. Changing graphics, layout or interactivity can be done anytime.
- Reduced chances for error: Physical data entry by employees is bound to lead to errors, which can lead to downtime, bugs and other problems. Rectifying such problems on data-driven websites is easy and this improves user experience.

2.3 UNDERLYING TECHNOLOGY LINKING DATABASES AND WEBSITES

The content of any website for a business or organization comprises data, including text, images, media and numerical values. Using a database to store this data is an efficient approach for many sites. If your site's data is stored in a database -- for example, using a database management system such as MySQL -- you may face the task of presenting the data within your Web pages. This process involves connecting to the database, querying it for data and presenting the data in HTML, often by using a server side scripting language such as PHP.

1. Prepare your database user account details. Database systems use accounts, with specific levels of access to each user. Your account details should include a username and password. Locate these details, if necessary copying them into a file. You will also need the name and location of your database. Find all of these details before you start coding. Your Web host should be able to help you with this information if you cannot find it.
2. Connect to your database. You will need to use one or more server side scripts to connect to your database. The following example code demonstrates making a database connection to a MySQL system within a PHP script: <?php mysql_connect("database_host", "username", "password"); ?> You will need to alter the host address, username and password to reflect your own account. The process for making a connection is similar for other database systems and programming languages.
3. Query your data. In most cases scripts use SQL (Structured Query Language) to retrieve specific sets of data from databases. These SQL queries can execute from inside a server side script. The following sample query demonstrates retrieving all records in a table named "Customers": SELECT * FROM Customers The following code demonstrates executing this query in PHP: $customer_result = mysql_query("SELECT * FROM Customers"); The variable contains the result data following the query.
4. Output your data. Once you retrieve the data from your database, you can present it within your site pages, which are structured in HTML markup. The following code demonstrates writing the query results into a page within HTML structures: while($customer_row = mysql_fetch_array($customer_result)) { echo " ".$customer_row['CustName']." "; } In this case the while loop iterates through each record in the "Customer" table, writing the value from a "CustName" field into the page as part of a paragraph element. You will need to alter the code to

reflect the fields in your database table and the HTML structures you want to display them within.

5. Test your script. Once you have your database connection script complete, or partially complete, upload it to your server to test it. If you encounter errors, check your database account details as well as the structure of your tables. Once you have established that you can connect to the database successfully in your script, you can build on the basic code to present your data to site users.

2.4 METHODS OF LINKING WEBPAGES TO BACK END PROPRIETARY APPLICATIONS

One of the central features of the Web is the ability for each Web page to offer connections to other Web pages and proprietary applications in the click of a button. **Proprietary application** is computer application for which the software's publisher or another person retains intellectual property rights usually copyright of the source code, but sometimes patent rights. Proprietary software often stores some of its data in file formats which are incompatible with other software, and may also communicate using protocols which are incompatible. Such formats and protocols may be restricted as trade secrets or subject to patents

There are a few ways that websites connect with one another, each with different legal implications for getting permission.

LINKING AND FRAMING

Two common ways websites connect to other sites are linking and framing.

1. Linking

Most often, a website will connect to another in the form of a link (also known as a “hypertext” link), a specially coded word or image that when clicked upon, will take a Web user to another Web page. A link can take the user to another page within the same site (an “internal link”), or to another site altogether (an “external link”).

You do not need permission for a regular word link to another website’s home page. If there is some concern over the link, most issues can be squared away by having the linked site sign a linking agreement that gives permission for your link.

Many copyright experts believe that deep linking (links that bypass a website’s home page) is not copyright infringement after all, the author of a novel can’t prevent readers from reading the end first if they so desire, so why should a website owner have the right to determine in what order a user can access a website? Some well-known websites such as Amazon.com welcome deep links. However, if a commercial website has no linking policy or says that deep links are not allowed, it’s wise to ask for permission before deep linking. Why? Because many websites even the listener-friendly Kaya FM have asserted rights against deep linkers under both copyright and trademark law principles.

International law is equally murky. For example, in 2002, a Danish court prevented a website from deep linking to a newspaper site. But in 2003, Germany weighed in on the issue when its federal court ruled that deep linking was not a violation of German copyright law. Subsequently, an Indian and a Danish court both separately ruled against the practice of deep linking in 2006.

2. Framing

Besides using external links, another way to connect from your website to other websites is by "framing." Framing is a lot like linking in that you code a word or image so that it will connect to another Web page when the user clicks on it. What makes framing different is that instead of taking the user to the linked website, the information from that website is imported into the original page and displayed in a special "frame." Technically, when you're viewing framed information your computer is connected to the site doing the framing not the site whose page appears in the frame.

Framing is generally unpopular with websites whose content is framed on another site (unless they have agreed to it). Websites that frame the content of other sites are often seen as stealing the other site's content.