
2ND EDITION
Babani Computer Books
How to create pages for the Web using HTML
J. Shelley

How to create pages for the Web using HTML

Titles by John Shelley

BP404	How to create pages for the Web using HTML (2nd edition)
BP453	How to search the World Wide Web efficiently
BP501	XHTML and CSS explained
BP517	Understanding the Internet
BP520	Fun Web pages with JavaScript (2nd Edition)

How to create pages for the Web using HTML

by

John Shelley

2nd Edition

BERNARD BABANI (publishing) LTD
THE GRAMPIANS
SHEPHERDS BUSH ROAD
LONDON W6 7NF
ENGLAND

www.babanibooks.com

PLEASE NOTE

Although every care has been taken with the production of this book to ensure that any projects, designs, modifications and/or programs, etc., contained herewith, operate in a correct and safe manner and also that any components specified are normally available in Great Britain, the Publishers and Author(s) do not accept responsibility in any way for the failure (including fault in design) of any project, design, modification or program to work correctly or to cause damage to any equipment that it may be connected to or used in conjunction with, or in respect of any other damage or injury that may be so caused, nor do the Publishers accept responsibility in any way for the failure to obtain specified components.

Notice is also given that if equipment that is still under warranty is modified in any way or used or connected with home-built equipment then that warranty may be void.

© 1996, © 1998 and 2000 BERNARD BABANI (publishing) LTD

First Published - October 1996
Revised and reprinted - February 1998
Reprinted - February 1999
Reprinted - January 2000
Second Edition October 2000
Reprinted – March 2001
Reprinted – May 2002

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

ISBN 0 85934 404 5

Cover Design by Gregor Arthur
Cover Illustration by Adam Willis
Printed and Bound in Great Britain by Guernsey Press

Preface

People creating web pages use the Hypertext Mark-up Language (HTML). Fortunately, it is an easy language to learn and use which accounts for the enormous growth of web pages over the past few years. Anyone can learn it in a matter of a few hours and create their own web page.

You may hear that HTML is doomed and that another language is going to dominate. Do not fear! This new language - eXtensible HTML, XHTML - is based on HTML. Far too many web pages already exist for them to be ditched in favour of some newcomer. Anyone who knows HTML can learn XHTML in an hour or so. So you will not be wasting your time. One of the major considerations of the new language was that it should be compatible with HTML. In fact, it uses all the same tags but does include a few new features. Chapter 12 discusses the future of Web authoring in some detail.

Web browsers, such as Netscape and Internet Explorer, can display information provided it has been written in HTML (or XHTML). That is why we need to learn these languages should we want to display information on the Web.

There are many programs around which will convert your text into HTML. Dreamweaver (from Macromedia) and FrontPage (from Microsoft) are two common ones. Word also has an additional program, called Internet Assistant, which converts ordinary Word documents into HTML. So do we need to learn HTML?

Yes! If you have to maintain existing pages or you want complete control over how your web pages look, then knowledge of HTML is a must. Once you know the language then these other programs can be used properly. People attending my HTML courses who have already learnt FrontPage or whatever, frequently say that

they wished they had attended the HTML course and *then* moved on to these other programs. They would have benefited much more since they would have understood exactly what was really going on.

Perhaps we should take some time now to explain the difference between the Internet and the World Wide Web (WWW). These terms are frequently confused.

The Internet is the physical means by which information is transmitted from one local area network (LAN) site to another, rather like the telephone system which connects one house to another. A wire from an office computer connects it to the organisation's local network computer (the Web server) or, a telephone connection from a home computer connects it to the Web server of a local Internet Service Provider (ISP).

In turn, these Web servers are connected to *routers* which route information from one local network to another via the telephone system, cables and satellites. They use a set of rules (*protocols*) agreed upon by all LANs using the Internet for the actual transmission of data. It is the ultimate example of a Wide Area Network (WAN). See *Other Titles of Interest* for reference texts.

At many local networks, information has been stored over the past decade on a vast range of topics in the form of databases, documents, programs, organisations' profiles, sound, film, video and images, etc. The entire collection of all this information stored on these local sites situated all over the world and the means whereby the information can be accessed (the Internet) is called the World Wide Web (WWW). To find and display any of this information on your own home/office computer screen a Web browser program is required and also an Internet connection.

Of course, we need to tell the program browser what we want to look for via a *search program* (such as YAHOO, AltaVista, Ask Jeeves or DogPile; there are hundreds

around) or to provide the browser with the name and address of a document (a URL, see page 50) given that we know exactly where it is held.

The first edition of this text was published in 1996. Much has happened since then and this second edition brings the reader up-to-date with the current versions of HTML. However, technology marches on and we discuss what life will be like in, say, five years time - 2005. HTML will still be used for several years to come and forms the basis for the latest breed of Web design tools. Indeed, you could still be writing all your web pages in pure HTML for many years to come. Chapter 12 discusses the future.

I would like to thank Mari-Elena Shelley for reading this text and for making many helpful comments prior to its publication.

About the Author

John Shelley took his Masters degree in Computing at Imperial College, London, where he has worked as a lecturer in the Centre for Computing Services for some twenty-five years.

He has been Chief Examiner since 1982 for the Oxford Local Delegacy in Computer Studies for their GCE O-level examinations, Senior Examiner for the SEG GCSE Computer Studies (now both defunct) and Chief Examiner for O-level Computer Studies for Overseas candidates.

He has written over a dozen other books on computing. He hopes this text will prove useful to those who want to learn HTML at a human level.

Trademarks

Microsoft, MS-DOS, Microsoft Windows, Internet Assistant, Internet Explorer, FrontPage, Outlook, Word are registered trademarks of Microsoft Corporation. The clipart used in some illustrations has been taken from the Clipart Gallery of Word 97.

Unix is a registered trademark of AT&T.

JavaScript, Dreamweaver, Eudora, Netscape, DogPile, AltaVista, AskJeeves are registered trademarks or copyrights of their relevant organisations.

Mac, Apple Macintosh, Sun are registered trademarks or copyrights of their relevant organisations.

References to the various URLs on the Internet are acknowledged here.

All other trademarks are the registered and legally protected trademarks of the companies who make the products. There is no intent to use the trademarks generically and readers should investigate ownership of a trademark before using it for any purpose.

Contents

1: Introduction to HTML

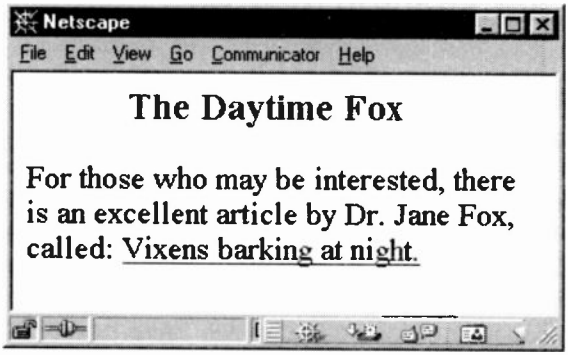
What is HTML?

As we all know today, when we want to look at something on the World Wide Web (WWW), we use our browser program, such as Internet Explorer, Netscape, Opera, etc. Browsers do not recognise Word documents, desk top publishing documents, spreadsheet files, or anything else. They would not know how to display such files. However, they do recognise documents which have been created in HTML. It is a language which all browsers understand. It is used by authors of Web pages to tell a browser how the content of their documents should be formatted - the size of heading, the colour of text, whether to use bold or italic, the positioning of images, etc. - in other words, how a Web document should be displayed.

HTML stands for ***H****yper****t****ext* ***M****ark-up* ***L****anguage*. The term *mark-up* comes from the printing industry. It refers to the days when a publishing editor would 'mark-up' an author's typed text so that the printer would know which words to make bold or italic, which size of type to use, say for chapter headings or footnotes, and so on.

Web authors must do this for themselves. Not only do they type in the content of the document but also the *mark-up* telling the browser how the text should look.

The original concept of the WWW was to allow people reading one document to click on a word or phrase which was blue and underlined and, lo and behold!, that reference would show up on the screen. No need to amble down to the local library or special libraries to find a copy of the reference. The blue and underlined text is called *hypertext*, it is ordinary text but has a link buried in the HTML source code which gives the Internet address of where the browser can find that page. We shall see how easy it is to create such 'hyped-up' text.



HTML is not a Programming Language

I am still surprised when I read books or articles on HTML to find that many expert authors claim HTML to be a programming language. It is nothing of the sort! If it were, then we would know exactly what would and what would not happen when our pages are displayed. We shall see later in this text that we will not really know exactly how our pages will look on every browser. But more of that when the time comes.

Programming languages are very exact in what they will do. HTML is not exact, in fact the display of many pages is un-predictable and will depend on which browser and which version is being used. A programming language, furthermore, is used to instruct a computer about what to do, such as perform calculations, control a washing machine, read information and react in one of several ways depending on what information it receives. You can do nothing like that with HTML. All you can do is to tell the browser how you would like your text to look.

In this sense it is a *text formatter*, but never in a month of Sundays, is it a programming language.

Text formatters pre-dated our current word processors. With the former, the text was mixed up with all the mark-up tags. This is how we approach our web pages when writing

HTML. We mix our text and mark-up tags together into what is known as the *source code* - the HTML document itself. This source code is read by a browser which will read the mark-up elements to see how the content (the text and images) is meant to be displayed. For example, here is some HTML source code:

```
Some normal text with <I> a phrase
in italic </I> and this part back to
the normal style.
```

After it has been read by the browser, it would be displayed like this in the browser's window:

> Some normal text with *a phrase in italic* and this part back to the normal style.

The `<I>` and the `</I>` are the mark-up tags. `<I>` tells the browser to move into italic mode; `</I>` tells the browser to turn off the italic style.

Frequently, when giving courses on HTML, participants are surprised to find how crude their HTML looks. We are familiar with word processors today. We type some text, select it and click a bold button, perhaps change the colour, size and type of the text and we *instantly* see the effect on the screen. HTML is not so sophisticated. You will not see how the text will look until it is displayed by a browser.

Where did HTML come from?

Back in the 1960s, IBM, then the largest manufacturer of computers, needed to transfer documents from one site to another. "Simple.", you would say. But not so in reality. IBM developed a General Mark-up Language (GML) to facilitate the exchange of documents across different operating systems and different application programs. There was a need for a common language, a *lingua franca*.

Documents which were marked up in GML and those operating systems and application programs which knew GML would be able to interpret the information correctly.

In the 1980s, microtechnology had advanced sufficiently for smaller firms and organisations to create their own networks for a fraction of what it would have cost a few years before. This was the so-called explosion in computing power. What was once the domain of those who could afford computing power - large pharmaceutical firms, banks, car manufacturers, government departments (Inland Revenue, Military) and, yes, universities (those were the heady days!) - was now in the reach of many smaller organisations. However, this led to a proliferation of different computer systems, each with its own 'language'. The Internet was growing rapidly in those days. Individual sites were linking together to form an International network (the Internet).

The International Standards Organisation (ISO) is a body with responsibilities across a far range of activities, one of which was for developing standards amongst the computer community. What common language would they choose for the Internet community? They adopted GML but called it SGML, the ***S*** standing for Standard.

By the early 1990s, with the advent of the World Wide Web, it was clear that a new language had to be developed to enable people to send and display documents via browsers over the Internet.

SGML was far too complex but it was used to define a simpler language - which came to be known as HTML. HTML, then, has roots dating back to ancestors in the 1960s. We tend to think of it as being all new and glossy, but it is not quite true.

Today, the body responsible for HTML and its future development is called the W3C (the WWW Consortium).

Versions of HTML

Version 0: was basically able to display text in a simple manner. The originator of the WWW, Tim Berners-Lee, so the myth goes, was surprised that the WWW was not

catching on too quickly. Text is fine for those who like to read - such as the academic community. But the vast majority wanted their text life to be spiced up.

Version 1: included the ability to display images. As micro-computers increased in power and memory size, the display of images became possible. It was still slow by today's standard but it was practical. As the myth goes, it was this which made the WWW catch on.

Version 2: included all the elements of level 1 plus the ability to use *forms*, see Chapter 9. With all those people reading our Web pages, would it not be a good idea to elicit some response from them. Forms were introduced to allow readers to return data to the originator of the Web page. For example:

- comments about the worthiness or otherwise of the page they have looked at
- filling out forms for orders
- registering for courses
- distance learning exercises
- questionnaires
- yea, even entering keywords for a search engine

Version 3: included the ability to create tables, frames, mathematical equations and figures. Tables lie behind the design of many web pages, see Chapter 10. Frames were the flavour of the month but today Web designers are more restrained in their use of frames although they can be very effective, see Chapter 11. Version 3 also permitted greater control over how text should look: colour, different typefaces, size, and so on.

Version 4: includes many of the multi-media features as well as the use of JavaScript and cascading style sheets (CSS). We discuss these in some detail in Chapter 12.

There is a degree of overlap between the versions but the above is a fair guide as to the differences between them.

Versions 1 and 2 were simple to learn and use; and all browsers implement their features. It was this very simplicity which led to the popularity of HTML. Since then certain software manufacturers, notably Microsoft and Netscape, have been tinkering away, adding non-standard features in the hope that their software would become adopted by most users. It has led to a great deal of frustration and annoyance for Web authors, but such is life. This book concentrates on level 2 and 3 since most Web browsers today implement the majority of their features.

The RFC 1866 Standard

You will find many references to RFC 1866. It is the standard for HTML version 2 as defined by the International Standards Organisation, (ISO), a body which defines standards used within the Computing Industry. (Today, it is the W3C - the WWW Consortium - which defines standards used on the WWW.)

There is nothing significant in the name. The number, 1866, happened to be the next Committee number, 1865 having already been used. RFC stands for **R**equests **F**or **C**omments. The ISO asked for comments about what should be included in the new HTML version 2. Some they rejected, others they kept. So, should you read somewhere that an HTML feature is RFC 1866 compliant, then you know that it is HTML 2. All browsers today are RFC 1866 compliant.

Unfortunately, the Committee never quite got around to defining exactly what version 3 should contain. There were reasons for this, but it gave Netscape and Microsoft the opportunity to go their separate ways, causing untold grief to Web authors. Our lovely Web page would be displayed beautifully by one browser but would not appear quite the same in another. The situation is being ameliorated today and Chapter 12 discusses the future development of Web technology in some detail.

Do I have to learn HTML?

There are several popular programs which will convert word processed documents into HTML or allow you to create professional looking web pages without the need to learn HTML. FrontPage, Dreamweaver and Internet Assistant are some examples.

They take time and effort to learn and use. If you are happy to accept whatever they offer and do not need to constantly make changes to your pages, they can be useful. But if you need to maintain other people's web pages or if you wish to make subtle changes to what these programs offer, then you will need to learn HTML. Indeed, once you know HTML, these other programs can become really useful. You can select part of what they offer and customise it to your exact needs.

(Frequently, some people who attend my HTML courses tell me that they wished they had begun learning HTML and then migrated to these other programs. It would then have made it easier to understand what these programs were actually doing.)

Another reason for learning HTML is that you may see a web page which catches your eye and you would like to know how it was written so that you can mimic the page by simply inserting your own text and images. Both Netscape and Internet Explorer (IE) allow you to view the source code of any page you look at on the Web. You click on the *View* menu and select *Page Source* in Netscape or *Source* in Internet Explorer.

IE will show the code via the Notepad program. Netscape will display its own source page which has the additional advantage of putting the HTML code in colour so that it is more easy to read the original source code.

However, be careful about copyright. A Web page has a certain design and its author(s) may not be too happy if you steal their web design. Always ask permission before you

lift people's designs off the WWW. If you know HTML, then it is simple to study the source code and find out how a page was constructed.

A final reason for learning HTML is that if you want to add JavaScript code to make your web pages more effective especially when using cascading style sheets, then an understanding of HTML is imperative, see Chapter 12.

Terms used

hypertext: a piece of text which is usually in blue and underlined which when clicked will display the Web page it references

mark-up: a printer's jargon for how text should look. The text is marked up with some code describing the format style to use

RFC 1866: the Committee which standardised Version 2 of HTML. All browsers must adhere to this standard

source code: a web author creates the content as well as the mark-up describing how the content should look. Together they form the HTML source code

typeface: the style of characters. This is Arial, this is Times New Roman, this is Comic Sans MS

W3C: the current name for the body which regulates standards for the WWW. It stands for the *World Wide Web Consortium.*

2: The HTML Language

The Basic Elements of HTML

HTML consists of two basic elements: *character entities* and *mark-up tags*. Since we shall not use character entities until Chapter 5, we shall not discuss them here. The mark-up tags are also called *elements* in many reference texts. We shall use both terms in this text.

Mark-up Tags

Mark-up tags are used to instruct a Web browser how to format the text. Each tag has a unique identifier enclosed in angle brackets - < >. 'B' is the tag identifier telling the browser to begin to display the text in bold. The </B> tells the browser to stop bolding. Thus:

```
Now begin to <B> bold this text. </B> Back to
normal.
```

would result in a Web browser displaying:

Now begin to **bold this text.** Back to normal.

<I> is the identifier for *italic* and <U> is the identifier for underline. These and many other tags have two parts, a *start* tag and an *end* tag. The ending tag is the same as the starting tag except that it includes a forward slash - /. Everything between the pair of tags will be displayed by the browser in bold, italic or whatever, and usually in the Times New Roman typeface. (I have used Arial in this text.) For example, the following HTML source code:

```
The following words are in <B>bold</B> and in
<I>italic</I> and <U>underlined</U>.
```

would be displayed by a Web browser as:

The following words are in **bold** and in *italic* and underlined.

Some tags have just one part, for example: <BR> and <HR>. The former is known as the *break* tag because it specifies that whatever text *follows* the tag must begin on

the next line. <HR> causes a horizontal rule (a line) to appear across the width of the screen. These elements are called *empty tags* because they do not contain any text to be formatted. Whereas, those tags which do format text are called *non-empty tags*. The latter are often referred to as *container* tags because they contain text to be formatted in some way. The *start* tag begins the desired format, the *end* tag turns off the format.

The tag name may be in uppercase, lowercase or a mixture: thus: `<hR>` `<HR>` and `<hr>` are all valid. However, strictly speaking, it is *not* correct to put spaces between the brackets or between the letters. Thus: `< HR>` or `< H R >` would not be recognised by some Web browsers.

A Complete HTML Document

An HTML document has two parts, a *head* and a *body*. Like human beings, a Web document should contain only *one* head and *one* body. Different browsers do different things when more than one body tag or head tag is met. Some ignore the 'error' and simply carry on, some will become confused and will not display anything which follows.

```
<HTML>
<HEAD>
<TITLE> Dotheboys Hall - School for young
Gentlemen </TITLE>
</HEAD>
<BODY>
Text and any images you want displayed....
</BODY>
</HTML>
```

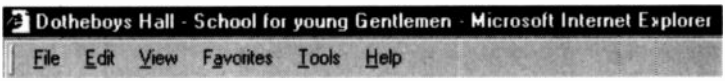
<HEAD> ... </HEAD>

The *head* contains general information about the document, mainly for the benefit of the browser, and is enclosed in the non-empty pair <HEAD> ... </HEAD>. What is contained in the <HEAD> is not displayed on the screen. For the moment we will use the head section to put in a title.

<TITLE> ... </TITLE>

Many browsers use the text within the <TITLE> tags to label the *display window* i.e. the blue title bar at the top of the window in which the Web document is displayed. The text also appears in the *Favorites* or *Bookmarks* lists should you save the address of that particular page.

It is pointless trying to format (change the font face and size of) the text within the <TITLE> tags, since the browser has its own built-in style for formatting titles.



The <TITLE> tags are also used by Web search-engines to collect keywords in Web document titles and add them to their database[1]. These databases are later used by search engines to find matches for keywords typed in by users of the search engines. Therefore, the text contained in the <TITLE> tags should be short yet sufficient to identify the document's content by use of keywords.

RFC 1866 specifies that all HTML documents *must* contain the <TITLE> tag and recommends that the text be less than 64 characters. Failure to include a <TITLE> tag will cause many of the latest browsers to display a blank page!

<BODY> ... </BODY>

The body part, enclosed between the tag-pair <BODY> ... </BODY>, contains the actual text and images which an author wishes to have displayed when someone calls up his/her Web page. This book concentrates upon what can be included in the BODY section.

<HTML> ... </HTML>

Strictly speaking, the entire document should be enclosed in the non-empty <HTML> ... </HTML> tags but this is currently optional. We shall not always include them in our

[1] See Appendix A, page 148, for more details.

examples for the sake of brevity. Some non-empty tags, such as the HTML and HEAD elements do not contain text to be formatted but must contain other tags. The HTML tag must contain a HEAD and a BODY tag; the HEAD tag must contain a TITLE tag, and so on.

A complete HTML document as shown in Exercise 1 is called the *source code*. Our browser program, such as Netscape, Internet Explorer (IE) or Opera, reads the source code and displays the text according to what mark-up tags have been included by the Web page's author.

Exercise 1: Creating a simple Web page
You may use any text editor or word processor to type the source code which includes not only the textual content but also the tags, but the resulting file must be saved as a *text only* file. Personally, I use Word for Windows; many others prefer Notepad. See page 146, for more details.

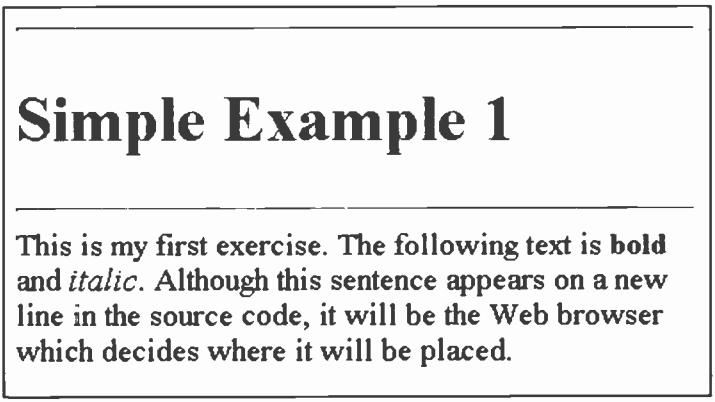
Call up your word processor and type in the following source code. Save the file as *text only* with a name and an `htm` or `html` *extension* in one of the folders on your hard disc. *(It would be a good idea to create a new folder specifically for your web files.)* Then use your Web browser to see what the document will look like on the Web. In Netscape or IE, this involves choosing *File*, then *Open File*; rummaging around to find the folder and then clicking on the name of the file.

```
<HTML>
<HEAD>
<TITLE>A Simple Example </TITLE>
</HEAD>
<BODY>
<HR>
<H1>Simple Example 1</H1>
<HR>
This is my first exercise. The following text is
<B>bold</B> and <I>italic</I>.
```

```
Although this sentence appears on a new line in
the source code, it will be the Web browser which
decides where it will be placed.
</BODY>
</HTML>
```

Notes:

1. You can see what the world will see when this Web page is called up.



2. Note the use of the empty <HR> tag to create lines before and after the heading - "Simple Example 1". The actual heading text is enclosed in a pair of level 1 heading tags - <H1>...</H1>. Headings are discussed below.

3. Exactly how many lines of text are displayed depends upon the size of the display window which a user has chosen. You can verify this by re-sizing your browser window.

4. The actual mark-up tags will obviously not be displayed .

5. When saving your file, you must include the `.htm` extension so that when it is opened by a browser, the extension tells the browser that it is a Web document and to display it as such. See Appendix A, page 146, if you want more details.

6. My source code has been laid out with line breaks so that it is easier for you (and me!) to read. I tend to 'mimic' how I want the displayed page to look like, as far as this is possible. However, the entire source code could be typed on one long line, indeed, this is effectively how it is received by the browser. It is the tags which then tell the browser how it must be displayed on the computer screen.

Likewise, I tend to use uppercase for tag identifiers although many prefer to use lowercase.

The Headings: <H1> - <H6>

Note the non-empty <H1> tag used in the code above. This denotes a Header 1 level. All the text contained between the starting tag <H1> and the closing tag </H1> will be in large, bold characters. There are six levels in all. Figure 2.1 shows examples of each. Note that some headings are in bold and others in italic. Usually, the text is in Times New Roman and a typical point size is also given. According to RFC 1866, <H2> must be displayed less prominently than <H1> but more prominently than <H3>. Likewise, <H3> must be displayed less prominently than <H2> but more prominently than <H4>; etc. The actual point sizes and typefaces are not defined and will be decided upon by whichever browser is being used. Today, most browsers display <H1> in 24 points (written as 24pt) rather than the 18pt size suggested by RFC 1866.

Headings have a pre-defined space to separate the heading from the text which follows. Usually this is a 12-point[2] line break *before* and *after* the heading for levels 1 to 4. Levels 5 & 6 have a break *after* the heading but not before, although many browsers ignore this and put a line space in before as well.

You may think that these headings are not very exciting and you are not alone in that thought. We will offer some variation later in Chapter 5 when we discuss *attributes*.

[2] See Jargon, page 16, for point sizes.

You should be aware that there are many different browsers and computers in use. On your browser, a document may look very fancy but on others which can only recognise simple HTML tags the page may look feeble.

Browsers are normally designed to ignore any tags which they cannot recognise either because the author mis-typed them or because the browsers cannot perform the required formatting, or because it is a tag which is recognised by a later version of the browser.

Level	Display format - actual size shown
1	<H1>**18pts Heading**</H1>
2	<H2>**16pts Heading**</H2>
3	<H3>*14pts Heading*</H3>
4	<H4>**12pts Heading**</H4>
5	<H5>*11pts Heading*</H5>
6	<H6>10pts Heading</H6>

Figure 2.1: RFC 1866 recommended point sizes

The <CENTER> tag

Headings are always left justified but you can centre a heading by enclosing it within the non-empty <CENTER> tags. Do note the American spelling of the tag name! This tag can contain almost any other HTML tag so that paragraphs, headings, images, etc., can be centred on the web page.

```
<HR>
<CENTER>
  <H3>My Main Heading </H3>
</CENTER>
This is a paragraph of text, left justified,
which follows the heading.
```

My Main Heading

This is a paragraph of text, left justified, which follows the heading.

Jargon

empty: refers to a tag which has no ending tag and consequently does not contain or format text.

identifier: a unique letter or group of characters which identifies which formatting style to apply to text, e.g. `<I>`.

non-empty: refers to a tag which has a start and an end tag. The pair contains text or other tags and is, therefore, sometimes called a *container* tag or element.

points: the size of a character. 72 points is an inch. 12 points (frequently written as 12pt) is a sixth of an inch, the standard size of text in many books. 10pt is commonly used for paperback books. This book has paragraph text in 10.5pt.

source code: a complete file containing both the content and the HTML tags. The source file is read by a browser and its content displayed as indicated by the HTML tags.

tags: also called elements. They tell the browser how to mark-up the contents of the Web page.

What you have learnt

How to create a complete Web page using mark-up tags. We have looked at how to bold, italicise and underline text. How to put in horizontal lines and what the various styles of the six headings look like.

A Web page can be created in any word processor or text editor but must be saved as a text-only file with a `.htm` extension. It can be saved on your hard disc and opened by your browser program so that you can see how it will look when it is finally stored on your Web server. This means that you can create Web documents without being connected to an ISP (Internet Service Provider).

A Web document has two main parts - a *head* and a *body*. It is what is placed in the *body* which will be displayed by the browser on to the computer screen. The head provides some basic information for the browser, such as what title to put on the top line of the window.

You must become aware that some browsers will apply their own styles to certain tags. For example, the *heading* tags will not necessarily be in the point size suggested by the RFC 1866. Furthermore, users are allowed to customise their browsers so that they can over-ride some of the styles an author may try to impose. In other words, how your browser will display your Web page may not be the same as the browser being used by other readers. There is nothing you can do about this. Simply be aware!

Elements & Tags

Consider this source code example:

```
<H3>A Level 3 Heading </H3>
```

The entire code is formally called an *element*.

This element consists of:

an *opening* or *start* tag:	`<H3>`
a *closing* or *end* tag:	`</H3>`
a tag *identifier:*	`H3`
content	`A Level 3 Heading`

The text between the opening and closing tags is called the *content* or sometimes the *contained text* and will be formatted and displayed by the browser according to the tag specified by the author.

You should try these tests at the end of the Chapters. They are meant to reinforce material covered, point out some common errors and extend the use of certain tags not covered in the Chapter.

Test

1. Four of the six heading tags put an automatic line space before *and* after the actual heading text. Which are they?

2. Two of the six heading tags put an automatic line space only after the actual heading text. Which are they?

3. What tag would you use to create a horizontal line across the whole width of the screen? Does this tag also have automatic spacing before and after the line?

4. *"The <TITLE> tag puts a title on to your Web page."* True or false?

5. If you accidentally put in two sets of <BODY> tags into your Web page, what would a browser do?

6. Which of the following tags are *empty* and which are *non-empty* tags:

```
<HEAD> <BR> <B> <TITLE> <HR> <H4> ?
```

7. What is wrong with the following?

```
<HEAD>The ABC plc Home Page</HEAD>
```

3: Formatting Tags

In this chapter, we shall look at those HTML tags which allow text to be formatted in various ways. <B> and <I> are two simple formatting tags, but there are many others. All of them are *non-empty* tags. We shall look at ten and divide them into the following groups:

- Bold: `<B>` & `<STRONG>`
- Italic: `<I>` `<EM>` & `<CITE>`
- Monospaced fonts: `<TT>` `<CODE>` `<KBD>` & `<SAMP>`
- Underlined: `<U>`

Bold - <B> & <STRONG>

```
<B> Text between BOLD tags.</B>
<STRONG> Text within STRONG tags.</STRONG>
```

Text between BOLD tags.
Text within STRONG tags.

So what is the difference, both are in a bold font? Usually no difference at all. Does that mean that I can use either tag and mix them up in the same sentence as demonstrated in the following?

```
The following is
<STRONG>a STRONG phrase</STRONG>
 and this is another phrase in bold
<B>and in the same sentence.</B>
```

The following is **a STRONG phrase** and this is another phrase in bold **and in the same sentence.**

This is not to be recommended. Text tagged with bold <B> should be displayed in bold by the browser. Text in <STRONG> will also appear in bold, but some of the latest browsers may decide to put <STRONG> text in a different

point size than text within a <B> tag. This would make the sentence above inconsistent.

The professional HTML author always remembers that there are many different browsers in use. To maintain consistency, the professional would ensure that both phrases are either tagged with <B> or with <STRONG>. Then, no matter which browser is used, both phrases will be displayed in the same way. Some browsers may not have a bold font at all, although this is very rare these days. It will have to decide how to distinguish, if at all, <B> text from <STRONG> text.

Italic - <I> <EM> & <CITE>

```
<I> Text will be in italics.</I>
<EM> Text will be emphasised.</EM>
<CITE> Text which refers to a citation.</CITE>
```

In most cases, all three would be rendered in italic and there would be no difference between them when displayed on the screen. But, there are some browsers which may well use different point sizes to distinguish between <EM> and <CITE> and <I> resulting in a strange sort of sentence if you were to choose to use all three in the same sentence for three different italic phrases.

```
There are three main parts to this object:
<I>the first part</I> followed by
<EM>a second part</EM>, and finally
<CITE> the third part.</CITE>
```

Clearly, the author intends all three phrases to have the same format. This will be the case with most browsers but not all. Your sentence which you are publishing for the whole world to read could look very odd on some browsers giving the reader a poor impression of its author.

According to RFC 1866, it states that <CITE> is used to indicate the title of a book or some other citation and is usually displayed in italic.

Monospaced Font Tags

The four 'different' types of *monospaced* tags are reminiscent of the erstwhile typewriter, where each character is given the same amount of space regardless of the shape of the character. The following is a good example: 'width'. The character 'i' is the thinnest character and 'w' is the widest, yet both are given the same (mono) horizontal space as each other. Whereas this 'width' is in a *proportional* font, where each character is given its required amount of horizontal spacing depending on its shape.

<TT>Text is in teletype font.</TT> Where a monospaced font is unavailable, an alternative representation may be used.

<CODE>Text which usually represents computer code.</CODE> It is intended for *short* words or phrases. It is recommended that <PRE> (see page 32) is used when there are multiple lines of text to be displayed in a monospaced font.

<KBD>Text which indicates what a user should type in.</KBD> This is commonly used in instructional manuals where a user is invited to type in some text at the keyboard. It is intended to be distinguishable from <CODE> so that what a user types in (input) is seen to be different from what a computer responds with (output). In most instances, browsers use the same font.

<SAMP>A sequence of literal characters.</SAMP> Text which should not change. It has a precise meaning for computer programmers.

One of the problems when beginning to learn HTML is the variety of tags which effectively end up looking the same.

In practice, many old hands at writing HTML documents use one or two of the above four tags (a sub-set). They keep to this sub-set and ignore the rest. For example, many would use <TT> and/or <CODE>, consistently of course, and

never use the other two. But we do need to know what the others signify just in case we come across them in someone else's source document.

In practice, I have rarely seen any of the above used today. But there was one occasion when I was looking at a page by a professional designer of Web pages. I saw this:

Please e-mail me, at this address: `john.smith@abc.co.uk`

I quite liked how the actual e-mail address stood out from the rest. On looking at the source code, I discovered that the address was contained within the non-empty <CODE> tags. In most browsers, you can view the source code of whatever web page is being displayed. Simply click on the *View* menu and choose *Source* (IE) or *Page Source* (Netscape).

Underline <U> ... </U>

This is the underline tag which underlines the enclosed text. Do you like to underline? Typographers, those concerned with the art and appearance of the printed word, regard the use of underlining as distinctly *naff* - crude. I agree whole-heartedly with them since the appearance of the under-line detracts from the clear reading of the text itself.[1] The whole purpose of providing written information is that is should be easy to read. However, there are many who love to underline and they are entitled to do so if they wish.

Having looked at most of the mark-up or formatting tags, it is time to turn our attention to those tags which put *structure* into a document. Incidentally, you may be beginning to think that what we are covering does not look much like the exciting Web pages we are familiar with. The text has no colour, has just one typeface (Times Roman), no pictures, all pretty boring really. Just hang on because all the more

[1] Hypertext is underlined for the benefit of simple browsers. It was their only means of being able to distinguish normal text from hypertext.

fancy things we can do require something called *attributes*. Once we meet these in Chapter 5, we can begin to make our Web pages look more interesting.

Summary of HTML Elements covered in Chapters 2 & 3

HTML tags	Comment
<B> ... </B>	bold
<BIG> ... </BIG>	makes text bigger than normal text - see Test #6
<BODY> ... </BODY>	contains the content of a Web document
 	break (force what follows onto the next line)
<CENTER> ... </CENTER>	almost anything can be placed within this tag to centre it on the page
<H1> ... </H1> ..... <H6> .... </H6>	The six heading tags
<HEAD> ... </HEAD>	the head tag for browser information (not to be confused with heading tags within the <BODY>)
<HR>	horizontal rule
<HTML> ... </HTML>	defining an HTML document
<I> ... </I>	italic
<SMALL> ... </SMALL>	makes text smaller than normal text - see Test #6
<STRONG> ... </STRONG>	equivalent to the <B> tag
_{...}	subscripts the contained text - see Test #6
^{...}	superscripts the contained text - see Test #6
<TITLE> ... </TITLE>	a mandatory title tag within the head
<EM> ... </EM>	equivalent to the <I> tag
<CITE> ... </CITE>	equivalent to the <I> tag
<U> ... </U>	the underline tag
<TT> ... </TT> <CODE> ... </CODE> <KBD> ... </KBD> <SAMPLE> ... </SAMPLE>	all should be rendered (i.e. displayed) in a monospaced font

Test

1. Why would experienced Web designers never mix <B> and <STRONG> tags within the same sentence?

2. What is a monospaced font? How does it differ from a proportional font?

3. What tags are available for italicising text?

4. How can you view the source code of a web page?

5. Would the following heading be centred?

```
<CENTRE> <H1>A Heading </H1> </CENTRE>
```

6. Try these.

```
<H2>Water - H<SUB>2</SUB>O - Everywhere<H2>
```

```
Here is a <BIG>big-word</BIG> and here is a
<SMALL>small-word</SMALL> and these are
normal text words.
```

```
<P>A Company<SUP>&#174;</SUP> registered
mark.

28<SUP>o</SUP>C degrees is 82<SUP>o</SUP>
Fahrenheit
```

The funny thing between the superscript tag is called a *character entity*. Chapter 5 and Appendix C cover these in more detail. The above is intended to introduce the subscript <SUB> and superscript <SUP> tags as well as a quick way of making text bigger or smaller - <BIG> <SMALL>. They are all non-empty tags. You should see the following:

Water - H_2O - Everywhere

Here is a big-word and here is a small-word and these are normal text words.

A Company® registered mark.

28^oC is 82^o Fahrenheit.

4: Structure

Web Screen versus Word Documents

Creating a web page for display on a monitor screen is totally different from creating pages to be printed out. They are different media. A printed page has top, bottom, left and right margins. A web page has only a top and a left margin. Its bottom margin depends on the total content of the page. Its right margin is dictated by the reader's window size which he/she may change at any time. There is no word wrap on a web page, therefore, the line length is dependent on the reader's whim. There is no concept of a page 1 and a page 2. There is just a vertical linear length and a browser will insert scroll bars if the content exceeds the current size of the window.

White Space

In a document created with a word processor, we make use of the Enter key to force text onto a new line; we use tab keys and perhaps put extra spaces between words. All these techniques create *white space*. But the browser will remove all of them. If your source code has three spaces between one word and the next, the browser will convert them into a single inter-word space. Tabs are ignored. So how do we structure our Web documents?

Some tags have got built-in structure associated with their use. For example, the <HR> tag automatically puts a blank line before and after the line; the <H1> to <H4> heading tags ensure that the heading text automatically begins on a new line with a blank line before and after.

Other tags do not have any built-in structure and, consequently, we have to make use of other tags to enforce structure. If you want some text to start on a new line, when typing an address for example, then you must put in a tag to say so. This is the purpose of the
 - break tag.

Thus, to create the following:

Heading Level 1

John Smith
e-mail: j.smith@abc.ac.uk
12th Dec 2000

I would type the following source code:

```
<H1>Heading Level 1</H1>
John Smith<BR>
e-mail:j.smith@abc.ac.uk<BR>
12th Dec 2000<BR>
```

Note:
There was no need to put a <BR> tag before "John Smith" since heading levels 1 - 4 have their own line breaks before and after the heading. Indeed, if I had done so, I would be increasing the gap between the heading and 'John Smith'.

**Line Break & Paragraph tags -
 & <P> ... </P>**
The empty <BR> tag forces whatever text *follows* the tag to begin on the next line. The non-empty <P> tag forces text to appear on a new line but allows for a blank line above the text. Thus:

```
Here is some text. <BR> This text will begin on
the next line. <P> This line begins a new
paragraph with a blank line above it.
```

would produce:

Here is some text.
This text will begin on the next line.

This line begins a new paragraph with a blank line above it.

Multiple <BR> tags will give additional line spacing but you should put a space between each tag since some browsers will act on the first and ignore the rest:
`<BR> <BR> <BR>` rather than: `<BR><BR><BR>`

The <P> tag has an associated end tag - </P>. The end tag is optional. It is often included by professionals to make their source code more readable. For the sake of brevity, it is frequently omitted in our examples.

The <ADDRESS> tag

The <ADDRESS> tag is used for addresses, signatures, authorship, etc., and is often placed at the start or end of the body of the document. It is conventional for web page authors to include their signature and perhaps any copyright notice. Typically, the text is rendered in italic and may be indented. The <BR> tag must be included when text is required on separate lines, since the <ADDRESS> tag has no built-in structure.

John Smith
e-mail: j.smith@ic.ac.uk
12th Dec. 2000

```
<ADDRESS>John Smith<BR>
e-mail: j.smith@ic.ac.uk<BR>
12th Dec. 2000<BR>
</ADDRESS>
```

The <BLOCKQUOTE> tag

The <BLOCKQUOTE> element is used to contain text quoted from another source and does have its own built-in structure. A typical *rendering* will provide a slight left indent, and a line space above and below the quotation.

As with the <ADDRESS> element, the <BR> tag must be used to force any following text to appear on separate lines.

The following quotation is taken from Shakespeare:

> There is a tide in the affairs of men,
> Which taken at full flood, leads on to fortune;
> Omitted, all the voyage of their life is drowned
> in shallows and in miseries.

Brutus, Julius Caesar Act IV Sc. III

```
<BODY>
The following quotation is taken from
Shakespeare:
<BLOCKQUOTE>
There is a tide in the affairs of men, <BR>
Which taken at full flood, leads on to
fortune;<BR>
Omitted, all the voyage of their life is drowned
<BR>
in shallows and in miseries.
</BLOCKQUOTE>
</BODY>
```

Tip: *You do not have to use BLOCKQUOTE for quotations. It could be used for indenting paragraphs and using the <P> tag to separate each one.*

Lists

There are several tags which can be used to create lists. The three most frequently used are: <UL> for bullets; <OL> for numbered lists; and, <DL> for hanging indents. With the exception of the hanging indent, the other two take the non-empty list item tag, <LI>, which is used to mark the start of each item in the list. The </LI> ending tag is optional.

Unordered List - <UL> ... </UL>

This is the unordered list consisting of a series of short lines, each marked with the <LI> element. Each line is usually marked by a round bullet or similar symbol and the text is indented from the symbol. If text wraps to the next line, it is aligned with the indent. It is called unordered to contrast it with the ordered <OL> tag which creates a numbered (ordered as 1,2,3 etc.) list. Suppose I want the following:

UL

Here is an unordered list, note the bullets:

- List item 1
- List item 2. Let's see what happens when this list item flows over to another line.
- List item 3

```
<BODY>
<H1>UL </H1>
Here is an unordered list, note the bullets:<BR>
<UL>
  <LI>List item 1
  <LI>List item 2. Let's see what happens when
this list item flows over to another line.
  <LI>List item 3
</UL>
</BODY>
```

Do not include a space between the <LI> tag and the actual text. If you do, the space may count as part of the text making the list look ragged at the left should you leave off a space on one of the other <LI> tags.

Ordered List - <OL> ... </OL>

This is similar to <UL> except that list items are numbered. In the following example, note how <OL> may be nested to produce a second level of indentation. Each list item in the second level is, again, marked with the <LI> element and must contain its own non-empty pair of <OL> tags.

```
<BODY>
<H3>OL </H3>
Here is an ordered list, note the numbering:<BR>
<OL>
  <LI>List item 1
  <LI>List item 2. Let's see what happens when
this list item flows over to another line.
      <ol>
         <li>substep 1
         <li>substep 2
      </ol>
  <LI>List item 3
</OL>
</BODY>
```

(The <UL> can also have other levels of indentation.) There is no significance in the use of lowercase for the indented list. It could also be in uppercase.

OL

Here is an ordered list, note the numbering:

1. List item 1
2. List item 2. Let's see what happens when this list item flows over to another line.
    1. substep 1
    2. substep 2
3. List item 3

Notes:

1. The use of indents and change of case helps to make the HTML source document easier to read.

2. With the <UL> list, most browsers show a different bullet style for the other levels of indentation. Again, what will be produced depends on what browser is being used.

3. You can specify the type of bullet and the numbering style to be used. This is done by adding an *attribute* to the <UL> and <OL> tags. See the table on page 47 in the next Chapter where we discuss attributes in detail.

The Definition List - <DL> ... </DL>

The definition list is used to create hanging lists. It does *not* take the <LI> element, but two other tags, <DT> ... </DT> and <DD> .. </DD>. It is intended for a list of named items, such as a *Glossary of Terms*, (the definition term - <DT>) and an accompanying paragraph of definition or explanation (the <DD>).

The entire list of items must be enclosed within the <DL> ... </DL> pair. The contents of a <DL> is a sequence of <DT> and <DD> pairs. The <DD> text is typically indented after its <DT>.

Most browsers assume the end of a <DT> or <DD> tag when another <DD> or <DT> or the ending definition list tag, </DL>, is encountered. In other words, both </DT> and </DD> are optional. Here is an example:

```
<HEAD>
<TITLE> Exercise for the DL, DTs and DDs
</TITLE>
</HEAD>
<BODY>
<H2>Glossary</H2>
<DL>
 <DT>The DL tag
 <DD>Stands for a Definition List
 <DT>The DT tag
 <DD> The term which is to be defined - usually a
short phrase.
 <DT>The DD tag
 <DD> The actual definition of the term in the DT
tag. Lengthy definitions extending over several
lines will align correctly.
</DL>
</BODY>
```

Glossary

The DL tag
: Stands for a Definition List

The DT tag
: The term which is to be defined - usually a short phrase.

The DD tag
: The actual definition of the term in the DT tag. Lengthy definitions extending over several lines will align correctly.

There are two more bulleted list tags, seldom used, but we need to know what they do in case we meet them in someone else's source code. They are the <DIR> and <MENU> tags and both behave identically to the <UL> tag. Instead of <UL> we use <MENU> or <DIR>. In the following we show the <MENU> tag, but try it out with <DIR> as well by substituting MENU for DIR.

<DIR> ... </DIR> & <MENU> ... </MENU>
These are similar to the <UL>, where each list item is bulleted. Most browsers, however, make no distinction between <MENU> <DIR> and <UL>

Here is an example of the <MENU> tag:

```
<HEAD>
<TITLE> Menu Example</TITLE>
</HEAD>

<BODY>
<H3>MENU</H3>

<MENU>
  <LI>List item 1
  <LI>List item 2. Let's see what happens when
this list item flows over to another line.
  <LI>List item 3
</MENU>

</BODY>
```

The Division Tag <DIV> ... </DIV>
This tag forces whatever text is contained between the opening and closing tags to be placed on a new line. It is similar in behaviour to the <BR> tag but is used more with *style-sheets* (see Chapter 12) rather than plain HTML. However, it can be used to create a list of contents without having to resort to the <UL> or <OL> tags. Try out the following:

```
<DIV>This is item 1 </DIV>
<DIV>This is item 2 </DIV>
<DIV>This is item 3 </DIV>
<DIV>This is item 4 </DIV>
```

There are two more elements we shall discuss in this chapter. The first is the <PRE> ... </PRE> tag and the second is how to insert comments within an HTML document.

Preformatted - <PRE>
This is suitable for text which needs to be displayed in a monospaced font. Unlike other tags, the <PRE> element

preserves line breaks and extra spaces. It is ideal for creating columns. According to RFC 1866, horizontal tabs within the text should be equal to 8 space-characters, however, it recommends that tabs should ***not*** be used but to use spaces instead.

```
<HEAD> <TITLE> The PRE tag </TITLE>
</HEAD>

<BODY>
<H2>Use of PRE tag </H2>
What follows is some text which has been
formatted in a monospaced font using PRE.

<PRE>
Column A          Column B
  Fred              Mary
 £34.56            £45.69
</PRE>

</BODY>
```

Use of PRE tag

What follows is some text which has been formatted in a monospaced font using PRE.

```
Column A          Column B
  Fred              Mary
 £34.56            £45.69
```

Text within the <PRE> tags is rendered (displayed) in a fixed-width typewriter font, usually Courier. It is the only mechanism, according to the strict standards of RFC 1866, for displaying tabular data or any other text which requires well-defined columns or absolute positions.

Any tag which implies some form of structure, such as headings, address, blockquote and lists, must *not* be used within the <PRE> text. The only tags which should be used are <B> and <I>. It can also take the anchor tag <A> for hypertext references, see Chapter 6.

It is rare today to see web pages using the <PRE> tag. It was employed in earlier days for browsers which could not display columns using the <TABLE> tag, see Chapter 10.

Adding Comments

Comments may be added to an HTML document for whatever purpose the author wishes, typically to explain to others what the author is trying to attempt, perhaps warnings about changes in staff or when a new pricing system is to be updated.

Comments begin with `<!--` and end with `-->` There must be no space between the exclamation mark '!' and the first `--`. All other spaces are treated as part of the comment. The entire comment is ignored by the browser and is seen *only* in the source document. Ideally, each comment should begin on a new line and avoid the use of special characters such as `< >&!`, since some older browsers will not interpret them correctly.

```
<HEAD><TITLE>Comments Example </TITLE></HEAD>
<BODY >
<H3>The ABC plc Home Page</H3>
<UL>
<LI>List of Staff Members
<!-- The current MD is leaving in Sept. 2001.
Update list with name of new MD. -->
<LI>Our Products
<!--    Price changes take place every Sept.
Update the price list. -->
<LI>Customer Support
</UL>
</BODY>
```

The ABC plc Home Page

- List of Staff Members
- Our Products
- Customer Support

What you have learnt

A web page is a scrollable screen, the printed word is typically on A4 size paper. The two media are different and require different approaches to how the content should be laid out.

Some tags have a built-in structure as part of their mechanism, others do not. Consequently, the <P> and the <BR> tags have to be used when an author needs to force text to begin on a new line.

We have looked at how to structure the content of a web page by using <P> and <BR> tags as well as using the built-in structure of *ordered* and *unordered* lists, the *blockquote* and the *hanging indent* tags. The six heading tags and the horizontal rule also have their own built-in structure.

Finally, we saw how to add comments to the HTML source code.

Collapsing White Space

Horizontal white space, such as tabs, extra spaces, and carriage returns, are 'collapsed' into a single space. Thus, twelve spaces between two words would result in a single space.

Vertical white space, that is lines before and after a tag (as with <H1>, <P>, <BLOCKQUOTE>, etc.) are also 'collapsed' into a single vertical space. In other words, when a <P> tag follows a previous <P> tag, there is only one line of white space, not two.

Summary of HTML Elements covered in Chapter 4

HTML tags	Comment
<ADDRESS>	renders text in italic but has no built-in structure
<BLOCKQUOTE>	provides blank lines above and below the contained text which is also left-indented
  & <P>	forces the following text onto the next line.

HTML tags	Comment
<P>	adds a blank line before the text which is forced onto a new line.
<DIR> <MENU>	alternative tags for creating bulleted lists
<DIV>	really used with style sheets. It behaves like the <P> tag, except that it does not give an automatic line space after the content.
<DL>	the Definition list used for creating hanging lists
<DT> <DD>	these are used with the <DL> tag for the term to be defined and the actual definition itself
<PRE>	the only tag which will guarantee the contained text will be in a monospaced font. White space is preserved
<UL> <OL>	the Unordered List (bullets) and the Ordered List (numbers)

Test

1. What built-in structure has the <BLOCKQUOTE> tag?

2. What would be the differences when using the <DIR>, the <MENU> and the <UL> tags to create bulleted lists?

3. Why could the addition of comments within your source code prove useful?

4. How could you get two paragraphs to be left-indented?

5. Which of the following tags have built-in structure and which have none?

<ADDRESS> <DIV> <BLOCKQUOTE> <UL> <MENU> <P>

So far, our web pages have not been especially exciting. In the next Chapter, we shall look at *attributes* since it is these which can add colour and different typefaces to our text.

5: Attributes

Attributes play a major role with some tags. They allow, for example, the choice of typeface, size, colour and the justification of text; the alignment of images; and, the creation of hypertext links. Not all tags can take *attributes* and for those that do, they are sometimes an optional extra.

What are attributes? They further define the use of the tag itself and must be placed in the starting tag. The attribute is separated from the tag ID by a space (not a comma). They usually take the form of: <ID *attributename* = "*somevalue*">

The value is usually placed in double quotes. Today, you should always use double quotes around the value even when it is not required by HTML. Why? Because the next version of HTML *always* requires values to be double quoted. Here is an example of the WIDTH attribute for the <HR> tag:

```
<HR WIDTH = "50%"> or <HR WIDTH="250">
```

Notice how the attribute is separated from the tag ID by a space, and that spaces may be present or absent around the equals symbol. Both examples are valid.

`HR`	the tag id
`WIDTH`	the attribute name
`"50%"` or `"250"`	the attribute's value

The WIDTH attribute specifies how wide the horizontal rule should be. When the value is a number it refers to the number of pixels to be used for the width of the rule. 100 pixels is about 1" (2.5cm). So the above "250" would be about 2½ inches.

The other value of the WIDTH attribute states that the rule must be 50% of the user's window. A browser window size of, say 4 inches, would result in the rule being 2 inches; the same rule would be 3 inches if the window was extended to 6 inches. Pixel values are always the same fixed, constant

width no matter what size window the user has chosen. A % value, on the other hand, will be adjusted to accommodate the window size. It will always take up 50% of the window size. Remember this when using other tags which can take a WIDTH attribute with pixel or percentage values.

The <FONT> ... </FONT> tag

Since we have introduced attributes, we can now mention the <FONT> .. `some text` .. </FONT> tag. As it stands the text contained within the tags would be displayed in the browser's normal font. But by adding some attributes, we can smarten up our text. The <FONT> tag can take the following attributes:

```
<FONT
    FACE = "comic sans ms, arial, sans serif"
    SIZE = "+2"
    COLOR = "red">
   Here is some text in Comic Sans,
   in a large font size and in red.
</FONT>
```

The FACE attribute

FACE refers to the typeface to be used. By default, most browsers use TIMES NEW ROMAN, a serif font. In the above code, we are asking the browser to display the text in *Comic Sans MS*. Why is it followed by `arial`? If the machine which the browser has been installed on has not got the Comic Sans typeface, then we ask it to use the Arial font. (Most computers have this font). But if it does not have Arial, then we ask the browser to choose any *sans serif* font which it has access to.

Note that each typeface is separated by a comma.

Serif and Sans Serif fonts

Look carefully at these two fonts.
One has little marks on the characters.
These are called *serifs*.

"The River Thames" in a serif font
"The River Thames" in a sans serif font

A sans serif font then, 'sans' being the French for 'without', has no serifs.

This book uses a sans serif font (*Arial*) for paragraphs and a serif font (*Courier New*) for the HTML source code.

The SIZE=2 or SIZE=+2 Attribute

There are two ways to specify the size of text. The first method is to provide a number in the range 1 - 7 as the value of the SIZE attribute. The number does not refer to pixels but to the size of text relative to the normal size. The normal, default base font is usually size=3. Thus:

`<FONT SIZE="5">` some text here `</FONT>`

would set the contained text in a size relatively 2 times larger than the normal font size; whereas:

`<FONT SIZE="1">` would be relatively smaller than the base font. The size is a scalar range defined by the browser. Thus, if the base font size is 3, there are two smaller sizes and four larger sizes to play with.

size=1 size=2 size=3 (normal) size=4

size=5 size=6 size=7

A second method is to use a *signed* value, -3 or +3 which would indicate a (+) larger or (-) smaller size based on the base font. Thus, if the base font were size 3:

`<FONT SIZE="-2">..your text ..</FONT>`

would set the text at a relative size two times smaller than the base font.

This feature can be used to create a large capital letter at the start of some important paragraph or heading:

```
<HEAD>
<TITLE>A Poignant View of Life </TITLE>
</HEAD>
```

```
<BODY >
<CENTER>
<H3> A Poignant Lesson from Life</H3>
</CENTER>
<BLOCKQUOTE>
<FONT SIZE="+3">T</FONT>o love is a pleasure,
a happiness which intoxicates, <BR>
to cease to love is ceasing to exist, <BR>
it is to have bought this sad truth: <BR>
   
<FONT SIZE="-1">
&quot; that innocence is a falsehood <BR>
       
love an art <BR>
       
and happiness a dream&quot;</FONT>
</BLOCKQUOTE>
from the <I>National Art Library Landing</I>
</BODY>
```

A Poignant Lesson from Life

To love is a pleasure, a happiness which intoxicates,
to cease to love is ceasing to exist,
it is to have bought this sad truth:

"that innocence is a falsehood
love an art
and happiness a dream"

from the *National Art Library Landing*

Add a splash of colour to the first letter via the color attribute (see below) and your page becomes more interesting:

```
<FONT SIZE="+3" COLOR="red">T</FONT>o love ...
```

The two strange things: ` ` (no break space) and `&quot;` (double quote) are examples of *character entities* which are discussed below. Each 'no-break-space' adds a

space (to indent the quoted "sad truth"); the 'double-quote' - `&quot;` - puts in a double quote.

The <BASEFONT> tag

You can change the base font from its default to another number. All text would now be set to the new size. For example, if you wanted to use 4 as the default font size, type:

```
<BODY> <BASEFONT SIZE="4">
```

This tag should come immediately after the BODY tag and before any text. FONT tags using a relative font size will now use the value set by the BASEFONT tag.

The COLOR Attribute

Several tags can take the *color* attribute. (Ah yes! do note the American spelling.) It takes either a name[1] or a hexadecimal value. However, since only a few names are recognised by all browsers, it is safer to use a hexadecimal value based on the three basic colours: Red, Green, Blue (RGB) - the three colours used by TV and PC monitors. Each colour has 256 variations from 0 - 255.

Each of the three colours takes a pair of hex numbers resulting in six numbers. Leading zeroes are required. FF is the hexadecimal number for 255. That is the purest value for each colour. Mixing the three pure colours of red, green and blue results in pure white. (Monitors are governed by the laws of Physics.) Thus: `COLOR="#FFFFFF"` is pure white and `COLOR="#000000"` is pure black. Every variation in between gives all the colours and shades of grey which a monitor can display.

Hexadecimal

The hexadecimal number system has sixteen digits: `0-15`, whereas our decimal system has ten: `0-9` and the binary system has two: `0-1`. All number systems include zero, hence `0-15` for the sixteen hexadecimal digits.

---

[1] See page 64 for some common colour names.

In hexadecimal, the digits 10 - 15 are given letters:

`A=10, B=11, C=12, D=13, E=14, F=15.`

Thus, D in hexadecimal refers to the equivalent decimal digit 13.

A two-digit hex number is sufficient for the 256 variations for each colour. 00 - FF is the range 0 - 255 in decimal. Thus each colour has a six-digit hex code.

Colour	Hex code RRGGBB
pure red	FF0000
pure green	00FF00
pure blue	0000FF
dark blue	090DEE

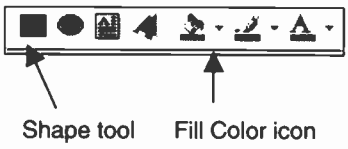
Thus:

```
<FONT COLOR="#FF0000"> text in red </FONT>
<FONT COLOR="#00FF00"> text in green </FONT>
<FONT COLOR="#0000FF"> text in blue </FONT>
```

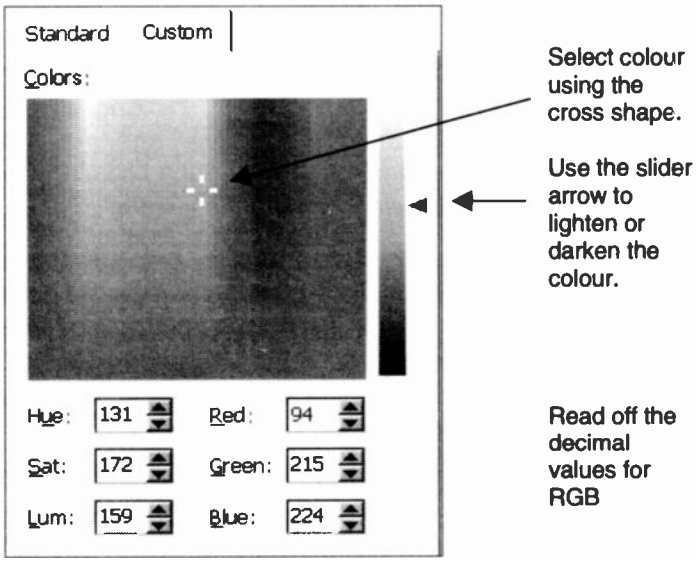
The value of the COLOR attribute should be preceded by a hash symbol (#) and enclosed in double quotes. Many browsers are tolerant should you forget the hash and the quotes, but this is not the correct approach.

How to work out the hex number for any colour

In Windows, it is very simple to discover the hexadecimal numbers for a particular colour. Use any application which permits you to create your own colours with the *Custom* tab of the colour palette. For example, in Word, Excel or PowerPoint, simply draw a rectangle or circle using the appropriate drawing toolbar shape. Then, use the down arrow on the '*Fill Color*' icon and choose *'More Fill Colors'* and then the *Custom* tab.



When you choose your colour, you will see the decimal numbers for the RGB.



Simply convert the decimal numbers into hexadecimal. This is easy in the *scientific* view of Windows *Calculator*. Type in the decimal number and click the Hex box. The decimal number will be converted to a hexadecimal number. These can then be used as the value for the COLOR attribute.

`<HR COLOR="#5ED7E0">` i.e.: 5E = 94; D7 = 215; E0 = 224

The preceding code will colour the rule in Internet Explorer but not in Netscape. (Welcome to the real world!)

<HR> tag

This may take the following attributes:

<HR

`ALIGN =`	`left|center|right`
`NOSHADE`	solid colour rather than groove takes no *value*
`SIZE =`	`10` vertical height of bar in pixels
`WIDTH =`	`10|80%` (10 = pixels; 80%=80% of screen)
`COLOR =`	`"#RRGGBB"` in hexadecimal, of course!

>

ALIGN: By default, rules are centred but using the ALIGN attribute, we can set the alignment to:

```
<HR ALIGN= "left|center|right">
```

Note the shorthand used above. The bar (|) means 'or' and is read as: align left or center (American spelling!) or right justification. It is a standard way of showing all the various values an attribute can take. Only one may be used. Justification of the horizontal rule makes sense only when it has a WIDTH attribute less than the window size.

The <H1> - <H6>, the <DIV> and the <P> tags may also take the ALIGN attribute! Therefore, you will now be able *left, center, right* your headings and your paragraphs. The <P> and <DIV> tags may also take the `ALIGN="justify"` attribute with the *justify* value. This will give full justification for a paragraph.

NOSHADE: Netscape grooves the rule into the background colour. IE provides a solid rule. So this attribute is only used for Netscape browsers when you want a solid line rather than a groove. I prefer the groove myself! Note that it does not take a value.

SIZE: The value must be a simple number which will refer to the size of the rule. A pixel value of 10 would result in a fairly thick rule since 1 or 2 are the usual sizes.

WIDTH: This may take pixel or percentage values and behave as mentioned earlier on page 37.

BODY colour

The <BODY> tag can also take an attribute which will provide a background colour for the entire page. It is the BGCOLOR attribute rather than COLOR which is used with the <HR> and the <FONT> tags. (Well, we have to be kept on our toes!)

```
<BODY BGCOLOR="#996633">
```

(The BODY tag can also be used to provide a background image for the entire web page. See page 73.)

Tip: *One way to pick up good colours is to view the source code of a Web page with a colour which you like. Make a note of the hexadecimal RGB and use it in your own Web pages.*

You should now try to use all these attributes to make your Web pages look more interesting.

Character Entities

Some symbols such as the < > " & /, have a particular meaning when typed into HTML source documents. If we wanted to display these characters, or other characters such as ½, ¼, we have to use *character entities.*

For example, should it be necessary to display:

"and <P> indicates the start of a new paragraph."

perhaps as part of a reference manual on HTML, typing <P> as it stands would simply be interpreted as the start of a new paragraph. It, therefore, becomes necessary to use character entities for the *less than* (<) and *greater than* (>) symbols, as follows:

```
and &lt;P&gt; indicates the start of a new paragraph.
```

A character entity begins with an ampersand symbol (&) immediately followed by the name of the entity and concludes with a semi-colon. A reference to a complete list of character entity names can be found in Appendix C, but here are some commonly used ones:

Ch. Entity	Meaning	Symbol
<	less than symbol	<
>	greater than	>
"	double quote mark	"
&	ampersand	&
	no break space	*'a space'*
¼	one fourth fraction	¼
½	half fraction	½
¾	three-fourths fraction	¾
á	a acute	á

The last four have no names, indeed, very few character entities have names. However, all of them have an ISO 8859-1 number.[2] This number consists of a *hash* sign (#) followed by the *decimal* number of the character. The hash symbol is sometimes referred to as the *pound* or the *number* sign.

The no-break-space is particularly useful when you want additional spaces to be inserted between words, to indent text, etc. This was used on page 40 to indent the quoted lines. It is also the method by which you can ensure that multiple <BR> tags will take effect:

```
<BR> <BR> <BR> 
```

```
<HEAD>
<TITLE> Character Entities </TITLE>
</HEAD>
<BODY>
Here are two hash symbols:  -- &#35; &#35; <BR>
The pound sterling &#163; symbol <BR>
The word<STRONG>format</STRONG>
is in &lt;STRONG&gt; tags
</BODY>
```

Here are two hash symbols: -- # #
The pound sterling £ symbol
The word **format** is in <STRONG> tags

Having looked at the basic HTML document, formatting tags, how to structure text, and been introduced to attributes and character entities it is now time to move on to creating *hypertext links.*

What you have learnt

Attributes enhance the use of many HTML tags. It is by using attributes that we can begin to make our Web pages look more attractive. We can use colour, different typefaces (with or without serifs), set the justification of paragraphs of

[2] See Appendix C for further details.

text, colour the background of a Web page, use different lengths, thicknesses and colours for rules. We also saw how we can convert any colour to its hexadecimal value.

Finally, we discussed character entities which allow us to display characters which we would not normally be able to.

Summary of HTML Elements covered in Chapter 5

HTML tag	Attributes
<BASEFONT>	SIZE - sets the font size for the web page text, usually size 3
<BODY>	BGCOLOR - background colour for page in "#rrggbb"
<FONT>	FACE - typeface SIZE - font size COLOR - text colour
<HR>	ALIGN - justification left|center|right NOSHADE - removes groove in Netscape - takes no value SIZE - height of line in pixels WIDTH - length of line, pixels or % COLOR - colour of line (IE only)
<P> & <DIV>	ALIGN - text justification left | center | right | justify The *justify* alignment makes sense only for paragraph text. The <P> tag does not take a color attribute, use <FONT> instead
<UL>	TYPE = disc|square|circle
<OL>	TYPE = 1|a|A|i|I

Test

1. What is the difference between setting the WIDTH attribute of the <HR> tag to a value of 50 or to 50%?

2. How could you set the font for all the text in a Web page to be two sizes larger than normal text with just one line of code?

3. Is anything wrong with this? `<BODY COLOR="pink">`

4. Write the code which will display the following onto a Web page:

"The *Diego Velázquez* exhibition is about ¼ of a mile along corridor D."

5. Try this out:

```
<CENTER>
<HR WIDTH="60%" COLOR="#090DEE"
    ALIGN="left" SIZE="3">
<FONT SIZE="+2"
      COLOR="red"
      FACE="comic sans ms">
Please Take Notice
</FONT>
<HR ALIGN="right" WIDTH="60%"
    COLOR="#996633" SIZE="5">
</CENTER>
```

6. How could you indent lines of text without resorting to the blockquote tag which would indent by a fixed amount as well as create blank lines above and below the blockquote text?

7. Why is the heading "`A Collection of Poems by Fred Bloggs`" in yellow? The author intended it to be in normal black typeface and thought that the closing FONT tag would achieve this by turning off the red and yellow FONT colours.

```
<H1><FONT COLOR="red">Poems on the
Underground</H1>
<FONT SIZE="2" COLOR="yellow">The London
Underground display poems. Here is one by Fred
Bloggs:
<FONT SIZE="-2" COLOR="green"> ...
       Fred's poem follows here ...
</FONT>

<H3>A Collection of Poems by Fred Bloggs</H3>
  ...  etc.  ...
```

8. How could you display small Roman numerals for your *ordered* lists? (Hint: look at the summary on page 47.)

6: Creating Hypertext Links

Before the advent of the WWW, we used to read printed papers which typically referred to some other source. If we wanted to read this other source, it entailed going to a library or requesting a copy by post. Using the WWW, however, we can simply click on a reference and it appears on our screens 'immediately'.

This facility is one of the main reasons for the interest in Web documents; they can contain *hypertext* as well as ordinary text. Hypertext is marked in a special way by Web browsers, typically coloured blue and underlined. When moving the mouse pointer over a piece of hypertext a little pointing hand appears rather like that in Windows when in the Help window. By clicking on the hypertext, another document appears. Buried in the HTML source code is the address of where the other document is stored. Hypertext is frequently called a *link* or *hyperlink*.

The <A> ... </A> tag

To create hypertext in HTML the <A> ... </A> container tag is used. Whatever is typed (is contained) between these tags becomes *hypertext*. The <A> stands for *anchor*, but by itself it is useless, it requires attributes before it can do anything useful. One of the main attributes is HREF - meaning a *hypertext reference* which provides the address of the document to be retrieved.

The HREF Attribute

When a hypertext link is clicked, what is next displayed may be:

- a document held on another server *anywhere* in the world
- a document held in the *same directory* as the current document
- or, some other position within the *same document*

The latter is useful when a document is long and a user may wish to move immediately to a particular part of that document without having to use the scroll bar.

It is the value of the HREF attribute of the anchor tag which provides the necessary information for the Web browser to find and display the referenced text.

For example, if I wanted the phrase 'Section 3' to become a hypertext link, it would have to be enclosed within the pair of <A> tags. The browser will then automatically make it blue and underlined.

There is more information in **Section 3** for those who are interested.

Here is the relevant code:

```
There is more information in
<A  HREF="http://www.ic.ac.uk/courses.htm">
Section 3
</A>
for those who are interested.
```

If a reader clicks this phrase, the browser will look at the value of the HREF attribute to find out the address of where the document is kept. It sends this address to its local web server which is connected to the Internet. The server requests a copy of the document from the site holding the web page. Once it has been received, the local web server passes it on to the browser which will now display the page.

The value of the HREF attribute is a URL. We shall discuss this now and then look at some examples.

When a Document is held at a Different Site
To point to documents held at a site other than your own local Web server, the link must contain a *complete* URL.

URL (Uniform Resource Locator)
A complete URL consists of:

- the *method* by which a document is accessed

- the unique address of the *site* (server) where the document is held
- the name of the document - *file name*

Example:

`http://www.ic.ac.uk/courses.htm`

`http` is the Web's own method for accessing and transferring documents between different Web servers. It stands for the *h*yper*t*ext *t*ransfer *p*rotocol. It is a set of rules (the *protocol*) used by network servers for transmitting data. See jargon at the end of the chapter for more details.

`://` is a separator marking off the transmission protocol from the rest of the URL

`www.ic.ac.uk` is the site address of where the document is stored, and refers to the Web server (*www*) at Imperial College (*ic*) which is part of the academic community (*ac*) in the United Kingdom (*uk*). Each site has its own unique Web site address, rather like each household has its own unique postal address or telephone number. Indeed, the site address *is* a set of four numbers each separated by a full stop. This is why you may sometimes see numbers rather than letters in some URLs: `http://123.45.06.78/`

Names are easier to type and to remember rather than numbers. The names used are converted to numbers by the Web server. It is rather like looking up a telephone number when you are given a name.

`courses.htm` is the actual file name of the Web page held in the server's storage discs.

Clicking on **Section 3** would cause the browser to look at the value of the HREF attribute, and request that document, using the *http* protocol, from the Web server which is holding that page. Once our browser has received a copy, it can then display the document onto our screen.

Thus, if someone at their office machine in, say, Houston, Texas, was reading a document containing:

Courses held at Imperial College, London
Courses held at Oxford University, England
Courses held at Birkbeck College, London

and they chose the courses at Imperial, the Web browser would contact the Web server at Imperial College, obtain a copy of `courses.htm` and display the document on the computer screen in Houston, Texas.

```
<A HREF=
   "http://www.ac.ic.uk/courses/course.htm">
    Courses </A>
    held at Imperial College, London
```

It should go without saying that the URL must be typed in correctly, just as we must correctly type *any* address or telephone number. This includes the correct name of the document. If the URL contains a mistake, the browser will not be able to retrieve the document. Case is significant in URLs.

You could, of course, simply display the URL as ordinary text and ask the reader to type it in the browser's address location box. But HTML offers hypertext as an alternative. When a reader clicks the hypertext, the end result is the equivalent of the reader typing the address into the browser's location box.

Document at the Same Site

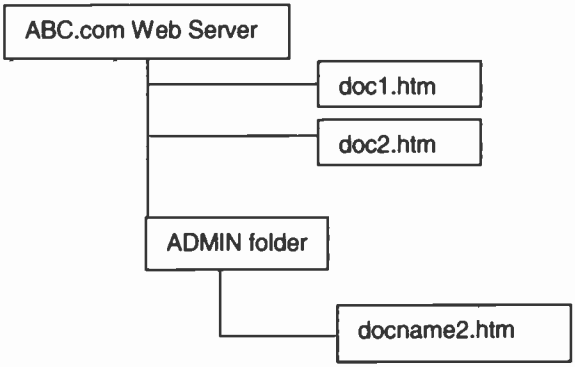
Should a document referenced by some hypertext reside in the same *directory* (folder) as the document currently on the screen, a shorter form of the URL can be used. Such links are called *partial* or *relative* addresses.

```
for more information <A HREF="doc2.htm">
click here. </A>
```

for more information click here.

The above code will assume that the document pointed to (`doc2.htm`) by the HREF attribute is another file in the

same directory and on the *same* server as the actual document you are looking at.



Sometimes, people on my HTML courses find this a little puzzling, so let us spell out what is happening.

Say the original document (`doc1.htm`) was displayed because the user:

- typed in the complete URL in a browser's location box
- used a search engine to find the document
- clicked a hypertext link in some other document

In all cases, the browser will have been given a complete URL, for example: `http://www.abc.com/doc1.htm`
It knows where it went in order to get the `doc1.htm` page.
If this page refers to another document, say `doc2.htm`, which is in the *same* folder as the `doc1.htm` page, the web author of `doc1.htm` needs to provide only a partial address:

```
<A HREF="doc2.htm"> a second document </A>
```

Here is a second document which you may like to see.

When a second document is clicked, the browser looks at the HREF's value and sees just the name of the document. It will fill in the missing bits because it knows where it found the original `doc1.htm`.

Thus:

http://www.abc.com/doc2.htm

The browser assumes that this new document is lying side by side in the same folder as the original doc1.htm.

Should the document be in a sub-directory of the current directory in which the document you are reading is stored, then that sub-directory would have to be included. Thus, if a document, say doc2.htm, references docname2.htm which is in a sub-directory named admin, then the following code would be used:

```
<A HREF="admin/docname2.htm"> ... </A>
```

Since the browser knows the address of where it found doc2.htm, it will fill in the partial address as follows;

http:/www.abc.com/admin/docname2.htm

These are called *relative* or *partial* addresses simply because the path specified to the other document is relative to the location (directory) of the current page being displayed by the browser.

Advantages of using Relative Links

One advantage of using relative links is that it reduces typing and the risk of mis-typing. But there is another and more important advantage. Let us suppose that one main document has links to several other documents. The main document and all the others are stored in the same folder on the Web server. Should it become necessary for the Web Manager to move all the documents to a different site or folder, all the relative links in the main document do not have to be updated. The relative links will become relative to the new location of the main document. Of course, the original URL of the main document will have to be changed so that users can locate its new location. But, once they have done so, all links in the main document become relative to its new location.

Moving within the Same Document

The following example is a single document which has information about the various courses a Centre may offer. The reader can immediately jump to the one he or she is interested in, or find details of how to get to the Centre, how to register, etc., by clicking on the relevant piece of hypertext (underlined in the example).

Training Centre Courses

Courses
Venue - How to Find Us
How to Register
Course fees

Courses:

Excel for Windows
HTML
JavaScript
Mac Operating System
Unix Operating System
Windows 2000
Word for Windows
.....

Venue

The Training Centre is situated in ..... etc. ...

Registration Details

To register for a course ..... etc. ...

Course Fees

Fees for the courses vary ..... etc. ...

Course Details

Excel for Windows ..... etc. ...
.....

To create a link in a document to some other point within the *same* document, the HREF must contain a pointer to that position and make use of the NAME attribute at that point.

Clicking on the hypertext 'Venue' will cause the browser to search for the marker venue elsewhere in the document and display what is at that point at the *top* of the window.

```
<A HREF="#venue">Venue</A> - How to Find Us
```

Note how the HREF attribute has no site address, no document name, just a hash (#) symbol followed by venue The name venue is invented by the author and case is significant!!

The hash symbol tells the browser that it has to look in the *same* document and not go off over the Internet to find another document. However, there must clearly be some point in the document which effectively says: "I am the place called venue." This is achieved by marking that point with *another* <A> pair of tags which has a NAME attribute with the value venue.

```
<A NAME="venue"> Here I am, you found me.</A>
```

Likewise, a click on the *How to Register* hypertext would jump to an <A> tag with a NAME attribute with the value H2Reg. Note that the # symbol is required in the HREF's value followed by the name of the place to go to; whereas the NAME attribute's value has only the name, no hash symbol.

```
<A HREF="#venue">Venue</A> - How to Find Us <BR>
<A HREF="#H2Reg">How to Register</A> <BR>
.....
.....
<H3><A NAME="venue">Venue</A></H3>
The Training Centre is situated in ..... etc.
... etc. ...
<H3><A NAME="H2Reg">Registration Details</A>
</H3>
To register for a course .... etc. ...
```

Browsers do not display the text contained within <A> tags with a NAME attribute in any special way. However, it is quite likely that the author may wish to format that text with, say, a level 3 heading. It is important that the pair of <A>

tags are contained *within* the <H3> format tags. Thus, if 'Registration Details' were formatted as:

```
<H3>Registration Details</H3>
```

and we now wanted to convert it into a hyperlink, then the <H3> tags should contain/enclose the pair of <A> tags thus:

```
<H3>
<A NAME="H2Reg">Registration Details </A>
</H3>
```

NOT:

```
<A  NAME="H2Reg">
<H3>Registration Details </H3>
</A>
```

Finally, since the text contained within <A> tags with a NAME attribute is not displayed in any special way, there is actually no need for any text to be placed within the pair. Thus, both the following are perfectly valid. It is the NAME attribute which forces the browser to locate that point and place it at the top of the browser window.

```
<H3><A NAME="come_here">Courses</A> </H3>
                    or
<A NAME="come_here"> </A> <H3> Courses</H3>
```

Hypertext & Courtesy

It is customary to enable users to return to the starting point of a long document without forcing them to use the scroll bar. Let us suppose that a user has clicked on a hypertext reference which calls up another part of the same document. Having read the information, the user may now wish to return to the beginning of the document (typically a list of contents, an index, etc.). The author should add a hypertext link back to that point in the document.

Likewise, an author may have several web documents referenced by one main document. It is useful to allow the reader to be able to return to the main document once one of the others has been digested. This is easily done by

putting in links in the other documents which return to the main document. This little courtesy saves the user having to click, perhaps several times, on the browser's *Go Back* button. Here is an example.

```
<HEAD>
<TITLE>'Document A' code - docA.htm </TITLE>
</HEAD>
<BODY>
<H2>Main Contents List</H2>
<UL>
    <LI><A HREF="intr-org.htm">
          Introduction to our Organisation</A>
    <LI><A HREF="staff.htm">
          Staff Members</A>
    <LI><A HREF="pgcrs.htm">
          Post Graduate Courses</A>
    <LI> .. etc ..
</UL>
</BODY>
```

In Document A - called docA.htm:

Main Contents List

- Introduction to our Organisation
- Staff Members
- Post Graduate Courses

Clicking on *'Post Graduate Courses'* in Document A would display Document B. Clicking on *'Return to the Main Contents List'* in Document B would force the browser to locate and re-display Document A.

In Document B - called pgcrs.htm

Post Graduate Courses

Here is a list of our post-graduate courses.

- Chemical Research
- Concrete Structures
- Environmental Engineering

Return to the Main Contents List

```
<HEAD>
<TITLE> 'Document B' code - pgcrs.htm</TITLE>
<!-- Post Graduate Courses -->
</HEAD>

<BODY>
<H2>Post Graduate Courses</H2>
Here is a list of our post-graduate courses.
....... etc ...
<A HREF="docA.htm">
Return to the Main Contents List</A>
</BODY>
```

How to do it within a single document:

```
<H2><A NAME="Top">Contents List</A></H2>
      Venue
      Courses
      Fees
.......  etc. .....
.......  etc. .....
<A HREF="#Top">Return to Top</A>
```

Contents List

.....

..... etc ... a really long document

.....

Return to Top

Clicking on *'Return to Top'* would cause the browser to search for the marker `'Top'` attached as the value of a NAME attribute within the same document. In other words, it would cause the browser to re-display the same document with the Contents List positioned at the top of the screen.

Port Numbers

`:1080` Sometimes, you may see what is called a *port* number attached to the web site address and preceded by a colon (:)

```
http://www.abc.com:1066/docname.htm
```

Fortunately, this is not something we have to worry about. That burden is part of the joy of being a Web Master or Web Manager. It is sometimes convenient for Web Masters to arrange that certain web documents are stored in a special area on their Web server's discs. Usually, most web pages would be accessed via the standard/default port, which is not usually given. But if one of the special documents is requested, then the URL must specify that the document can be found only by going through a particular port. These are simply numbers but must be specified if given in a URL. It will be the Web Master who decides whether our documents require a port number or not. Our only concern is when we need to publish the URL of our Web page to the public at large, in which case it would have to be included.

Other Protocols

`http` indicates the Web protocol, but many other protocols can be used as well.

`file:` This method causes the browser to load a file from the locally accessible disc system and is frequently used to preview Web pages being developed on a computer that has a browser but no server, just a hard disc, such as our home or office PCs.

`mailto:` E-mail Form (and see Chapter 9). The e-mail address follows the colon (:).

`news:` USENET newsgroups - the group name follows the colon.

`ftp:` file transport protocol. An earlier Internet tool used for transferring files. It is still possible to see URLs using this protocol.

`gopher:` the gopher protocol was used to search for information on the Web in the days before browsers. It was an early type of search engine.

`telnet:` an earlier Internet tool for accessing resources at another site. Essentially it allowed outsiders to poke about in folders to see what was there.

Sending e-mails via Mailto:

It is common for Web documents to include e-mail addresses which when clicked will call up the browser's e-mail program so that a reader can send a message to the author. It saves the reader from having to call up his/her own e-mail program and then having to return to the browser to carry on reading.

```
If you have any comments why not
<A HREF= "mailto:j.smith@abc.co.uk">
e-mail me? </A> I would be most grateful.
```

The above would look like:

If you have any comments why not e-mail me? I would be most grateful.

A browser's e-mail window has the typical: TO: FROM: SUBJECT: boxes and a bigger box for the message. The reader fills in the message and clicks on the SEND button all without leaving the Web page he/she is reading.

Note that both the MAILTO: and the NEWS: have just one colon after the protocol, not the usual `://` as with the *http* protocol. This is simply because both systems existed long before browsers and knew nothing (nor cared) about the syntax of the *http* protocol.

```
Allergies <A HREF="news:alt.med.allergy">
try this.</A>
```

would produce:

Allergies try this.

Jargon

protocol: protocol is a set of rules recognised by two or more parties. In our case, the parties would be Web servers. English grammar is a protocol, a set of rules for communicating in English. If both parties use the correct rules, they can understand what the other is saying.

case: refers to style of letters, either in *UPPERCASE* or *lowercase*

What you have learnt

We have seen how to create hyperlinks to other documents via the <A> tag and its HREF attribute. The value of the HREF attribute provides the address of where the document is held.

The URL may be a full URL complete with the protocol to be used, the site address, perhaps a port number and sub-folders, and the document name. If no document name is given, each site has a special page (the *home page*) which it will send out. This will frequently contain information about what the site has to offer.

```
<A HREF="http://www.abc.com/">
```

If the document is in the same folder as the one currently being viewed, only a partial URL need be used. Just the name of the document. `<A HREF="doc2.htm">`

If the document is in a subfolder of the document currently being viewed, then the name of that subfolder must also be included.

```
<A HREF="subfoldername/doc2.htm">
```

We have also learnt how to move about within the same document by including the section marker (the #) followed by some invented name.

```
<A HREF="#top">Return to top. </A>
```

In this case, somewhere else in the same document, there must be *another* <A> tag with a NAME attribute whose value is the same as the invented name.

```
<A NAME="top">
```

We looked at some of the other protocols which browsers can use to send or retrieve data which are not HTML documents, such as e-mail and news groups.

Test

1. What attributes can the anchor tag take?

2. What does this mean:

```
http://www.abc.ac.uk/crses/course.htm#courses ?
```

3. How many mistakes can you find in the following?

```
<A HREF="#gothere>click to go there </A>
.............
<A NAME="#Gothere">Here I am </A>
```

4. How could you get your readers to send you an e-mail message without having to leave their browsers?

5. When do you have to include port numbers in a URL?

Colour Names

There are sixteen common colour names which most browsers can recognise. Those marked with an asterisk are not always accepted by some browsers. It would be safer to use their hexadecimal values.

Colour name	Hex value	Colour name	Hex value
black	"#000000"	green	"#00FF00"
silver	"#C0C0C0"	lime	"#008000"
gray (US spelling)	"#808080"	olive	"#808000"
white	"#FFFFFF"	yellow	"#FFFF00"
maroon	"#800000"	navy	"#000080"
red	"#FF0000"	blue	"#0000FF"
purple	"#800080"	teal	"#008080"
fuchsia	"#FF00FF"	aqua	"#00FFFF"
* lightblue	"#CCFFFF"	* lightgreen	"#8DF78D"
* lightyellow	"#F1EF95"	* pink	"#FF99CC"

Tip for those grey days:
If you subtract equal amounts from RGB you get shades of grey. Thus: `aaaaaa`, `bbbbbb`, etc.

or: `111111`, `222222`, etc.

or even: `e1e1e1`, `a5a5a5`, etc.

But `556677` is not equal amounts and thus a colour results!

7: Putting Images onto Web Pages

Adding an image or picture can brighten up your web pages. Indeed, it was only once images were added to version 2 of HTML that the Web really began to be used. But, it should be remembered that each image is a separate file and it takes time for the browser to locate it over the Internet. It then takes time to display that file on the web page. The bigger the image, the longer it takes to retrieve and display.

Most Web browsers can display images in GIF (`.gif`), JPEG (`.jpeg` or `.jpg`) and PNG (`.png`) formats. We have more to say about these formats later.

The <IMG> tag

An image can be:

- inserted by itself anywhere in a document - just for decorative purposes
- inserted along with text
- inserted as a hypertext link so that when clicked it will display another document or image

We shall look at these in order.

Inserting an Image just for Decoration

The <IMG> tag has no end tag, that is, it is empty. However, it must contain at least one attribute, SRC (source), telling the browser where to find the image. When images for a web page are created, they are stored as separate image files. Therefore, a document which incorporates an image must contain not only the image *tag* but also the *location* of where the image file is stored. The SRC is in fact an image equivalent of the HREF attribute and the syntax is identical to that of HREF. Here is an example:

```
<HEAD>
<TITLE>Putting in an image. </TITLE>
</HEAD>
<BODY>
<CENTER><H1>Text with an Image</H1></CENTER>
<FONT SIZE="+3">H</FONT>ere we see the Great
Crested Geek bird at home in a London marsh.
<P>
<IMG SRC="GCgeek.gif">
</P>
Note the fine crest feathers. </BODY>
```

Text with an Image

Here we see the Great Crested Geek bird at home in a London marsh.

Note the fine crest feathers.

To ensure the image is separated from any other text in the document it has been enclosed within the non-empty paragraph tag. Adding the end tag </P> forces the image to be a stand-alone image. For once it is not optional.

The basic syntax is: `<IMG SRC="url">`. If the image file is in the same location as the document then *relative* links may be used. If the file is in a sub-directory of the current directory then the sub-directory needs to be included. If you tend to work with several images, it is sometimes more convenient to store them in a sub-folder, for example: `images`. This means that your image files and your web pages are kept separate.

(We are talking about your computer's hard disc where you store your web pages and images whilst you are developing your web pages. Eventually, you will have to pass all the files to your Web Manager who will decide exactly where they will be stored on the web server.)

```
<IMG SRC="images/imagename.gif">
```

If the image file is stored at a different site, then the full URL must be given:

```
<IMG
SRC="http://www.ic.ac.uk/images/oddface.gif">
```

To Align Text with an Image

In the above example, the image was displayed in its own 'surrounding', text above and below but it is possible for text to flow on the same line as the image, for example:

Where to pitch your tent!

Most camping sites provide information about their facilities.

```
<IMG SRC="tent.gif">
Where to pitch your tent!<P>
Most camping sites provide information about
their facilities.
```

Note how the text is aligned at the bottom of the image - this is the default position. See the ALIGN attribute below for other placements. The <P> tag forces the following text below the image.

Making an Image a Hyperlink

To turn an image into a hyperlink, simply include the IMG tag *within* the non-empty anchor tag, <A>, thus:

```
<A  HREF="url">  <IMG SRC="url">  </A>
```

Note that we have two URLs: one for the image tag so that the image can be found and loaded into the *current* document. A second for the <A> tag so that when and if the image is clicked, the appropriate URL can be located and displayed. The URL referenced by the HREF attribute can be any valid URL, such as another image, a normal web document, a news group reference or an e-mail message.

A slight adjustment could make an image and text hyperlinks.

```
<A HREF="url"> <IMG SRC="url"> or this text</A>
```

If you remove the <IMG> tag from the above, we are left with just the text being a hyperlink. Including the <IMG> tag within the anchor element and we have made the image as well as the text a hyperlink to the document referenced in the HREF-url, either of which could be clicked to obtain the file.

There are two more attributes associated with the <IMG> tag, the ALT and the ALIGN.

Other IMG attributes

ALT="*text*"

Not everyone wishes to display images. In order to speed up display, some browsers allow an option to turn off '*the display of images*' - this is not an uncommon ploy especially when people have slow connections. So what do these people see?

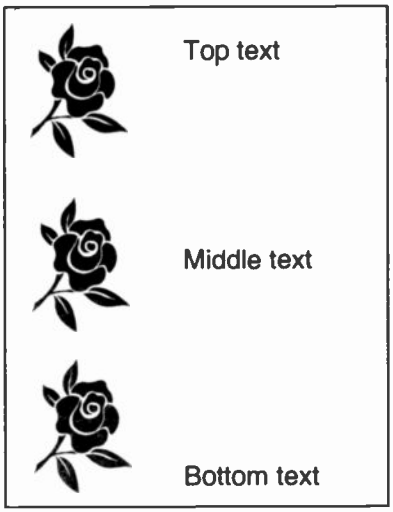
They will see whatever text you have typed between the double quotes in the ALT (Alternative) attribute, thus:

```
<IMG SRC="http://www.ic.ac.uk/images/mydog.gif"
ALT="My name is Fritz.">
```

It is a matter of courtesy to let those in the above situations know what they are missing. So in place of the image, they see your text. (Some browsers actually display this text until the image file is finally loaded into the document or when the mouse is moved over the image. In the above example, when someone moves the mouse over the image of my dog, my dog's name will be displayed. It is recommended always to use ALT.)

ALIGN="*position*"

In HTML version 2, the *position* values are *top*, *bottom* or *middle*. It specifies the alignment of any accompanying text. The following shows the results of using all three.



Note that with short phrases the text aligns as desired. However, if the text runs into a sentence, the rest of the text which does not fit on the line will automatically drop to the *bottom* of the image. This means quite a gap between the lines of text as shown on the next page. Note how ugly it looks.

To overcome this problem HTML version 3 included two new values for the ALIGN attribute RIGHT & LEFT. These

attributes refer to the position of the *image* (not the text as with the version 2 values). Images were always left justified in version 2.

If the text is too long,

it continues at the bottom!

```
<IMG SRC="/images/oddface.gif" ALIGN="right">
Quite a long sentence but which will wrap to the
left of the image since we have positioned the
image on the right.
```

Quite a long sentence but which will wrap to the left of the image since we have positioned the image on the right.

Warning when using `ALIGN=LEFT|RIGHT`

If you have tried out either of these values, you may have noticed that *all* subsequent text and even small images will wrap on the left or right of the image. I am frequently asked on courses: "How can I get just some of the text wrapped and force the rest to appear below the image?"

There are two methods. The first is to use a TABLE but this is not discussed until Chapter 10 in the Test number 7 on page 116. For the time being, we can use the <BR> tag with the CLEAR attribute value set to `"all"`.

`<BR CLEAR="all">` This will ensure that all text which follows moves below the image. You can also use a `left` and a `right` value which will work only when the image `align` attribute is the same. It is safer to use the `all` value, then it does not matter how the image is aligned.

The BORDER Attribute

If you have inserted an image and made it a hyperlink, you may have noticed that a ghastly blue border surrounds the image. Browsers do that to show that it is a hyperlink, in the same way that they turn text into blue and underline it. Worse still is the purple border that some browsers put on a visited link. It is simple to remove this border. Use the BORDER attribute of the IMG tag and set its value to 0. This tells the browser to push the image up to the border, effectively preventing the blue border from becoming visible.

```
<IMG SRC="rose.gif" BORDER="0">
```

Resizing Images

There are two more attributes we can discuss, the WIDTH and the HEIGHT. These specify the size in pixels of the image's width and height.

```
<IMG SRC="url"  WIDTH = "190" HEIGHT = "256">
```

If you want to resize the image, the simplest method is to enter just one of the attributes. Your browser will automatically work out the correct ratio for the other. So if I wanted the above to be just 150 pixels high, I would simply enter: `<IMG SRC="url" HEIGHT = "150">`

You should also be aware that enlarging or reducing an image will inevitably result in loss of the original quality. So your best approach would be to decide on the exact image size within whatever image processing program you are familiar with, for example Adobe PhotoShop.

Using Internet Assistant

If you open a file with an `htm` extension in Word which has Internet Assistant added into it, the page is displayed in a manner very similar to a browser. Indeed, Internet Assistant is a browser, but in Office 97 it has limited capabilities. In Internet Assistant, the image can be clicked to show re-sizing handles. Using one of the corner handles and dragging will enlarge or reduce the image

without destroying its proportions. When you click the *View* menu you will see *HTML Source*. By selecting this, you are returned to the source code where you will see that Internet Assistant has adjusted both attributes.

Warning: *If you select HTML Source, Internet Assistant adds a few <META> tags (see page 148) to your original source. It also has the annoying habit of reformatting the line lengths of your original code by converting to full justification. The Office 2000 version adds even more of its own proprietary code.*

If you wish to resize an image within Internet Assistant using the re-sizing handels, make certain that the image is not a hyperlink, otherwise when you click It to reveal the handles, Internet Assistant will interpret the mouse click as an "open this web page" and will fetch whatever the HREF's attribute value points to. You will be unable to get to the re-sizing handles.

Image Formats

There are many different formats for storing digital images. Each has advantages and disadvantages. At the present time, web browsers are able to recognise just a few of these formats. Thus, any image to be inserted in a document should conform to one of these formats: GIF, JPEG or PNG.

GIF

Graphics Interchange Format - with a `.gif` filename extension. It is a format which all graphical web browsers can recognise. It is especially useful if the graphical image is a logo, an icon or a banner, where there is little variation in colour detail. It can store black-and-white, greyscale and colour images, although it is limited to 256 colours per image. It is also useful when *transparent* images are required. This allows any background colour on the web page to show through the transparent areas. It also allows for *interlacing*. Usually, images are built up pixel line by pixel line starting at the top and working to the bottom. Interlacing is a technique whereby groups of

lines are displayed, interspersed throughout the image, so that the entire image is seen in more and more detail giving the viewer an overall 'picture' of the image from the outset.

JPEG

Joint Photographic Experts Group is a format especially designed for storing photographic images. Its file extension is `.jpeg` or `.jpg`. It has a 24-bit colour depth and should be used when a high level of colour and detail must be preserved, for example with photographs.

Generally, speaking, JPEG format is better than GIF for photographic images. The quality is better and through its more sophisticated compression techniques the resulting files are smaller than an equivalent GIF version.

PNG

The last format we shall mention is the Portable Network Graphic (`.png`). Like GIF, it allows for transparency, interlacing and image compression. It has better colour quality than GIF so why have I given up on using `png` images. The problem is that Navigator and IE tend to show the colours differently. The same image often looks darker in IE than in Netscape, to the extent that the detail becomes blurred. But you must make up your own mind.

Adding an Image to a web page

Via the BODY tag, we can apply an image as a background to the entire page, using the BACKGROUND attribute whose value is an image file.

```
<BODY BACKGROUND="image.gif">
```

We saw on page 44 that the BODY tag can also take the BGCOLOR attribute. What happens when both are used? The BACKGROUND image will always take precedence over the BGCOLOR. In other words, the image will sit on top of the coloured background. However, if the image has any transparent areas or is oval in shape within a transparent box, then the background colour will be able to show through.

If the image is smaller than the web page, browsers will *tile* the image so that it fills the entire window. (One page I liked had a blue coloured bar about one inch wide on the left and the rest was white. There was a pleasing ragged edge between the blue and the white bands. The image was ½" in height and 10" wide. This image was used as the value of the BACKGROUND attribute in the BODY tag. Being tiled, the overall effect was a blue left-hand bar with white for the remainder. These are very simple to create in image processing programs and can be saved as `gif` files.)

Loading Images

a) The browser first retrieves the document itself and displays the text and leaves spaces for any <IMG> elements. It then makes further connections to the Internet via the IMG's SRC-url, which provides the address of the image files. Thus a document with five images requires six separate connections to the server, one for the original document, then five more trips for each image file. This clearly involves time before the complete document and its five images can be fully displayed.

b) Images which have been scanned using one of the many available scanners are frequently saved in a non-web format. I usually save all my scanned images in TIFF (Tagged Image File Format) because it is an industry standard and saves the image in high quality. All image processing programs can open a TIFF image. These programs can then be used to touch up the image and save it in GIF, JPG or PNG.

c) When an image is loaded, most browsers will store the image in a *cache* memory (a part of your hard disc). If the image is required again, perhaps because it is repeated on the same page, for example a repeated image used as a bullet - see page 109, or perhaps because the user re-

sizes the window thus forcing the browser to reload (re-display) the entire page again, the image can be retrieved from the cache memory rather than the browser having to retrieve another copy over the Internet. This clearly speeds up the display of images.

d) Whenever practical, always include the *width* and the *height* attributes. When they are present, the browser will allocate the correct space for the image and use the *alt* attribute's *value* until the image is loaded. If the size attributes are not present, some space is allocated for the image but since the actual size is not known, the entire page will have to be repainted once the image arrives. In other words, you slow down the eventual page display when not present but can speed up the display when the *width* and *height* attributes are present.

e) Finally, although it was quite obvious once it had been pointed out to me, I had not given much thought to exactly what the WIDTH and HEIGHT attributes of the IMG tag actually do. Say you have an original image file of 50K bytes. You reduce its size exactly by half using the attributes above. This does not speed up the delivery of the original file. The original 50K will still have to travel over the Internet. Then once it has arrived, the browser will fit the original into the space you have given. In other words, the original will not be reduced to 25K until it is processed by the browser, so it will still take the same amount of time to arrive.

What you have learnt

You have learnt how to incorporate images into your web pages; how to wrap text around the image; how to convert an image into a hyperlink.

You should now be able make your web pages much more attractive.

Summary of HTML Elements covered in Chapters 6 - 7

HTML tag	Attributes
<A>	HREF - "url" NAME - place to move to in a document
<BODY>	BACKGROUND - tile an image on web page BGCOLOR - colour web page background
 	CLEAR - left | right | all. It is safer to use *all*
<IMG>	ALIGN - left | right ALT - alternative text to display for image BORDER - numeric value HEIGHT - in pixels height of image WIDTH - in pixels width of image SRC - "url" of where image is stored

Test

1. What IMG attribute is required to display an image within a web page?

2. If the image you want loaded is stored at some external site what would happen if that site moved the image to some other folder without telling you?

3. Let us say that you see a web page with your organisation's logo on it. You would like to use it on your own web pages and have been given permission to do so. How would you obtain a copy of that logo?

4. You see a picture on someone else's web site and would like to use it for your own web page. The answer to the above question tells you how simple it is to obtain that image. But could you be in breach of copyright?

5. You have made an image a hyperlink, but are now dismayed to find that a horrible blue line runs around the entire image box. It looks even worse after it is clicked because now the browser puts a *purple* border around the image. How can you get rid of the border?

6. How can you tell whether you have Internet Assistant in your Word program?

8: Image Hot Spots

Hot Spots

A single image can have *hot spots* added so that when one of the spots is clicked, an anchor tag HREF attribute can be invoked. This feature is known as an *Image Map* because we create areas of hot spots on the image, just like a geographical map which has areas: regions, counties, countries, etc.

An image map consists of three components:

1. the actual image itself which is linked to a <MAP> tag
2. the non-empty <MAP> tag which contains the areas to become the hot spots
3. the empty <AREA> tags which define each hot spot

Get the Image

It is an ordinary HTML <IMG> tag:

```
<IMG SRC="myimage.gif">
```

except that we need to include a link to the *map* to be used. For this we need a new attribute in the <IMG> tag, namely: USEMAP.

```
<IMG SRC="myimage.gif" USEMAP="#mymap">
<MAP NAME="mymap">
.....
</MAP>
```

This now links the <IMG> tag to the relevant <MAP> tag whose NAME attribute value is the same as that in the <IMG> tag . The USEMAP value has a section marker (#) - just like that in an anchor tag's NAME attribute.

```
<A HREF="#venue">Venue</A>
```

The <MAP> & <AREA> tags

We shall now use these two new tags to create three hot spots, as shown below.

```
<MAP NAME="mymap">
<AREA HREF="Mid-England.htm"
      ALT="Middle England"
      SHAPE="rect"
      COORDS="175,68,231,95">

<AREA HREF="excel1-3.jpg"
      ALT="An Excel spreadsheet example"
      SHAPE="circle"
      COORDS="75,37,22">

<AREA HREF="UK.gif"
      ALT="Enlarged map of the UK"
      SHAPE="poly"
      COORDS="301,42,268,95,330,118,383,93,
              330,90,301,42">
</MAP>
```

The non empty <MAP> tag contains a number of empty <AREA> tags. Each one defines the position of the *hotspot* in the image via its COORDS attribute. Do not be put off by all the numbers, they are very easy to work out!

The SHAPE attribute defines one of three shapes: a circle, a rectangle and a polygon. You choose which shape best suits the area on the image which is to become the hot spot.

The ALT attribute is used to provide text when a user points over a hot spot. Annoyingly, Netscape (version 4.5) does not show these ALT values or when it does only one is shown. IE behaves in a more civilised manner.

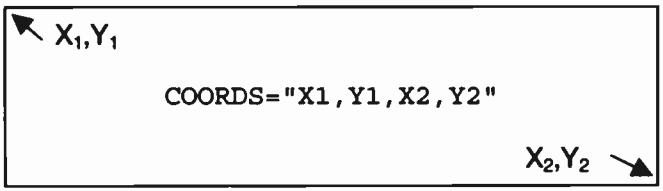
The HREF attribute provides the URL of the file to display when the hot spot is clicked.

The COORDS attribute defines the area on the image which is to become the hot spot. This is done by entering pairs of *x,y* co-ordinates, each separated by a comma. The numbers must be in pixels and the actual number of pairs depends upon the shape of the area.

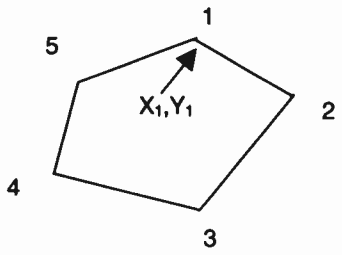
The 'difficult' thing is how to work out the co-ordinates. That is what we show below. There are little programs which can be used to create the HTML for the AREA tag, but they have to be bought (or downloaded over the Internet) and then learnt. If you have an image processing program, such as PhotoShop, Image Composer, Photo Editor, Paint, it is easy to do it yourself, so why bother?

Co-ordinates to create a rectangle:
Two pairs of co-ordinates are required. The upper-left X_1,Y_1 and the lower-right X_2,Y_2, that is four numbers in total, each separated by a comma.

Read off the ***pixel*** values from your image program. In PhotoShop jot down the *x,y* co-ordinates as shown in the *Information Window*. In Photo Editor, Image Composer and many other similar programs, the information is found on the *status bar line*. (See examples below.)

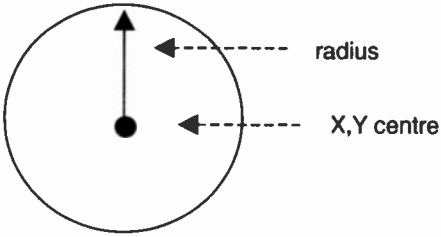


Co-ordinates to create a Polygon:
You need to read off the *x,y* co-ordinates for each corner of the polygon:
corner 1: X(1), Y(1),
corner 2: X(2), Y(2),
corner 3: X(3), Y(3)
... etc.



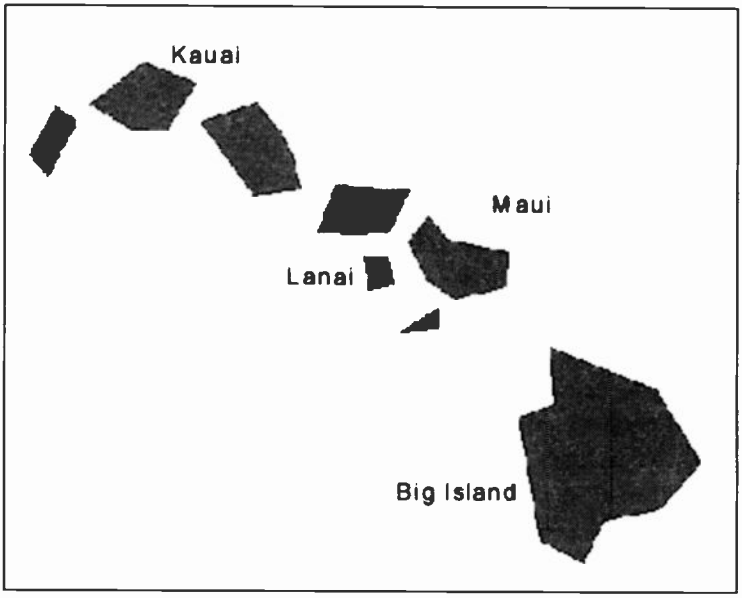
This five sided polygon requires 5 pairs, that is 10 numbers in total.

Co-ordinates to create a circle:
A circle is defined by its *centre* point - a single pair of *x,y* co-ordinates and its *radius*, resulting in three numbers.

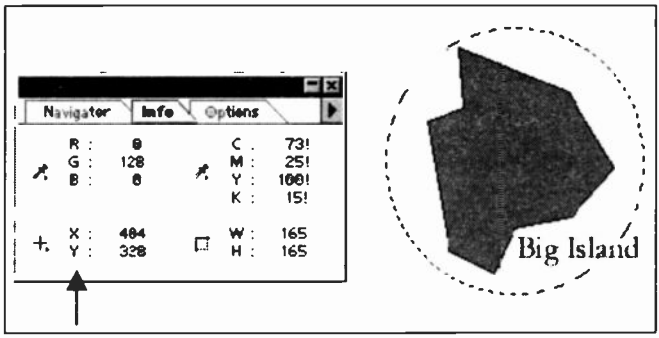


The radius is the distance from the centre point to the outer edge of the circle.

How to read co-ordinates from image programs
The following is a map of the Hawaii islands. We wish to make three of the Islands hot spots.



Circles - PhotoShop



In PhotoShop 5, a simple way is to draw a circle, using the Shift key to make it a perfect circle. Point your cursor in the centre of the circle and read off the x,y co-ordinates, 404, 328 in the above example.

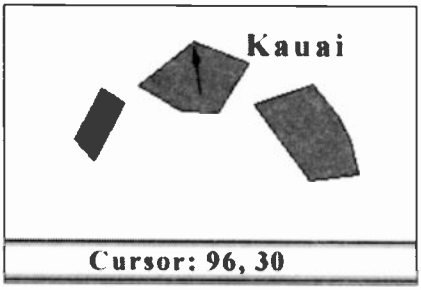
Notice the WH (width / height), they should be the same for circles: 165 in the above illustration. The radius will be half the height. Divide by 2 to find the radius:
`165/2 = 82.5`, say 82 pixels.

In the above, `"COORDS=404,328,82"` would define the co-ordinates for the circle. Although Big Island is an ideal polygon shape, I think a circle would be better for the user. Everything enclosed within the circle will become the hot spot, including the white area outside Big Island.

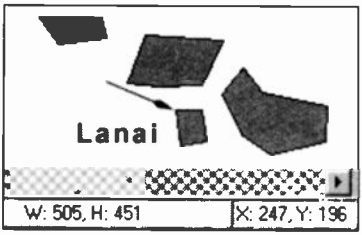
The same principle applies to all image processing programs. When the cursor is placed over a part of an image, the x,y co-ordinates are always shown somewhere.

Polygon - Photo Editor

The next illustration is for the polygon shape for Kauai Island. Note the cursor pointing to the top position and the Cursor information (the X,Y) at the bottom. This is how Photo Editor displays the X,Y co-ordinates. Corner 1, then, is 96, 30. Repeat for each of the remaining corners.



Rectangle - Image Composer
Here is a rectangle for Lanai Island using Image Composer. The arrow points to the top-left corner and shows the *x,y* co-ordinates for the first pair: 247,196. Incidentally, the W H shows the size in pixels of the entire image.

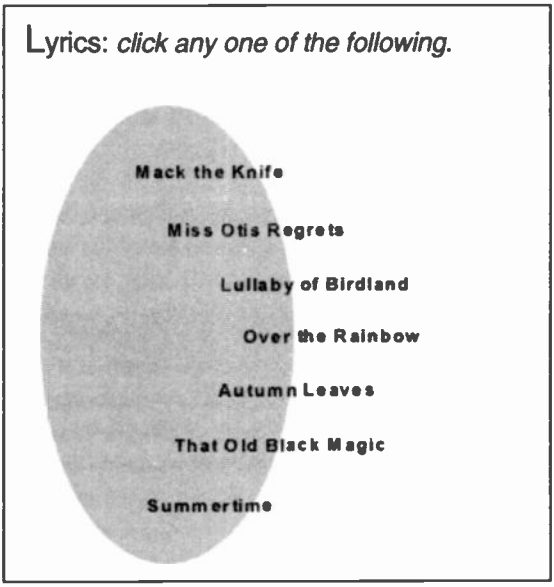


All image programs will show the X,Y co-ordinates of where your cursor is currently fixed on an image.

Tip: *If you have made a selection, say by using one of the selection tools (ellipse, circle, rectangle, etc.), most image programs will display the size of the selection as width and height values.*

What you have learnt
We have seen how to create hot spot areas on an image so that when one is clicked, the relevant HREF-url will be displayed.

Using hot spots is quite a good way of creating a list of contents rather than using the <UL> tag to create a bulleted list. Here is a simple example to give you the basic idea. It is one image file including all the text.



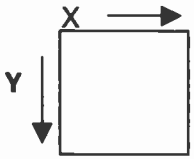
Each song is a rectangular hot spot which when clicked could display the lyrics of the song via the HREF attribute of the AREA tag.

Summary of HTML Elements covered in Chapter 8

HTML tag	Attributes
<IMG>	USEMAP = "#somename" use the named map for hot-spots areas
<MAP>	NAME = "somename" the value associates this MAP with a particular image's USEMAP attribute
<AREA>	HREF ALT SHAPE = poly | circle | rect COORDS *(TARGET - target the link to a separate frame, see Chapter 11)*

X-Y Co-ordinates

At school, we learnt that the 'x' value starts at zero and increases as we move to the right. Unlike school, the 'y' value starts at the top left and increases as we move down.



9: Forms

HTML provides a means whereby information may be typed into a form on a web page by a reader, collected by the browser and sent off to some other location. In other words, you can solicit user input from your web page readers. Here are some instances where forms may be useful:

- registering for courses
- sending credit card details (!) for ordering goods
- forwarding abstracts of research papers for storage in a public database
- forwarding comments about the worthiness of some article to its author
- details for an on-line questionnaire
- soliciting information to be sent to a specific e-mail address
- typing in keywords for a *search engine*

The <FORM> tag is the HTML method for getting readers to fill-in forms. The basic idea is that the user is invited to type information into text boxes and/or click various other types of boxes within the form. These boxes may be blank text areas into which the user types in text, or clicking radio buttons, check boxes and pop-up menus (for selecting items from a list), a submit button and usually a reset button. When the submit button is clicked the data, which the user has entered, is collected together by the browser and sent off to its destination. The reset button clears any data entered by the user so that he or she may start again.

There is a problem. Each form has to have a program specifically designed to process the set of data each time the form is filled in.

These form-programs may be written in any programming language, typically Java, C, C++; or, scripts (a Unix term for server programs) written in *perl*, *tcl*. It is beyond the scope of this text to delve into programming, and, in any case,

writing such programs requires knowledge of the web server being used to process the data. However, there is one type of form submission which does *not* need any such programming knowledge. This is what we discuss in this Chapter. (For those who are interested, there is a brief discussion at the end of this Chapter, about the Common Gateway Interface (CGI) which is the standard mechanism for communication between *http* servers and the forms' programs, frequently called *gateway programs*.)

There is an example of a form in Fig 9.1, where readers are invited to register for a JavaScript course. It could apply to many other applications such as ordering goods, tax returns, job applications, indeed any form which we may need to fill in.

How do we construct a form?

We use the <FORM> tag. The FORM tag has a starting and an ending tag and the starting tag must contain at least two attributes: METHOD and ACTION. The entire form is enclosed within the pair of form-tags. For the form to be of any use, it must also contain at least one <INPUT> tag and frequently a non-empty <TEXTAREA> tag as well as a *submit* button so that the data can be sent off to the e-mail address. There may be several FORMs in a document, but FORMs cannot be nested. We shall discuss the <FORM> tag first and its two main attributes before looking at the other tags.

```
<BODY>
.........
<FORM METHOD=POST
      ACTION="mailto:myname@someplace">
<INPUT attributes>
<TEXTAREA attributes> </TEXTAREA>
<INPUT TYPE="submit">
</FORM>
.......
</BODY>
```

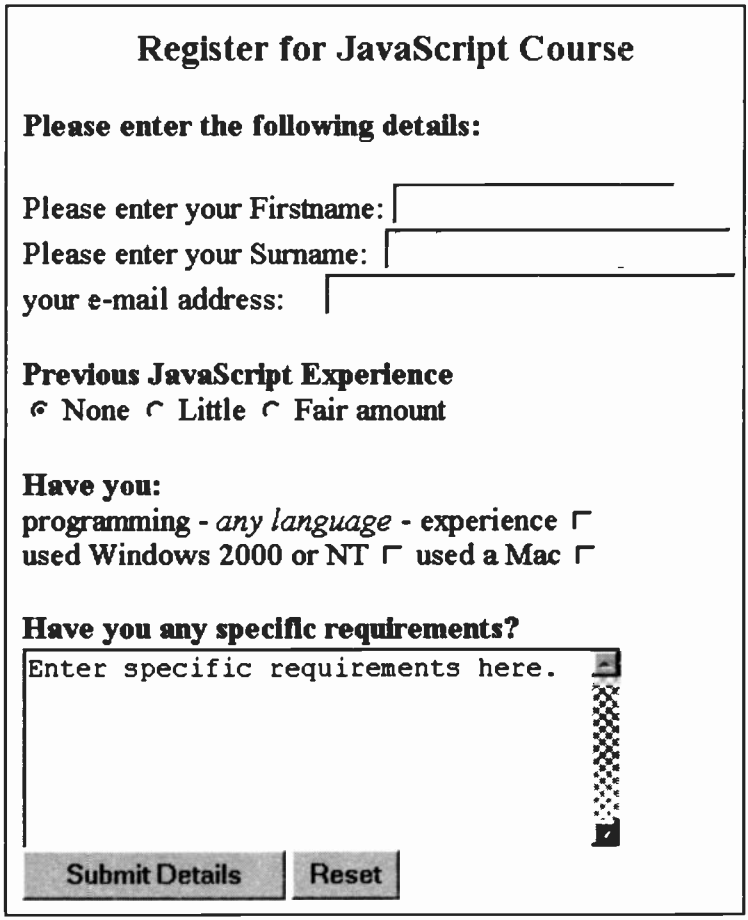


Figure 9.1

The METHOD attribute

The METHOD attribute can take one of two values, either GET or POST. The GET appends the data filled in by the user to the form-document's URL. It is called a *query* URL and consists of the form-document's URL, a question mark and the data itself. This is passed to the program which will process the data specified in an ACTION attribute. That

program uses the data to do any number of things such as searching or updating a database.

You may have noticed this sort of thing in your browser's *Address* box *after* you have clicked on a *Search Engine's* search button. Here, I used *AskJeeves.com* to search for "Mack the Knife". The following is the data string sent to the server program:

```
http://askjeeves.com/main/askjeeves.asp?ask=Mack+
the+Knife&origin=&site_name=Jeeves&metasearch=yes
&IMAGE1.x=14&IMAGE1.y=11
```

The POST value (the one we need) determines that the data is sent (posted) either to a gateway program for processing, or to an e-mail address. But instead of being appended to the URL, the data is sent as a block. GET is the default value, that is why we need to specify POST.

Exactly what the differences are between GET and POST are beyond the scope of this book and is the subject of specialist texts. Should you have to design a form for your web page, you can either have the information sent directly to your own e-mail address or get someone in your organisation to write a program for you. That person will tell you whether to use the GET or POST value for the METHOD attribute and supply the name and address of the program which will process the submitted data.

The ACTION attribute

Once the submit button is clicked, the data typed in by the user into the form will be sent to the URL specified in the ACTION value.[1] For example:

[1] We now have three attributes which take a URL; the HREF, the SRC and here the ACTION.

```
<FORM METHOD=POST
ACTION="http://www.abc.edu/cgi-bin/gateprog.xyz">
                    or:
<FORM METHOD=POST
      ACTION="mailto:fred@xyz.ac.uk">
```

The first example would send the data to `gateprog.xyz` stored in the gateway programs' directory (`/cgi-bin/`) at the web site `www.abc.edu` for processing via the `cgi` protocol. The `gateprog.xyz` program would have to exist.

The second example would send the data to the e-mail address referenced in the ACTION attribute, using the `mailto:` protocol. Note the double quotes around each of the URLs.

<INPUT> - an empty tag

This tag specifies the type of input-element to use within the form. It is an empty element and requires at least two attributes: TYPE and NAME.

`TYPE="display"`

The value of the TYPE attribute specifies which type of input mechanism to display in a form. The following lists the various input mechanisms allowed.

TYPE values	Purpose
`"text"`	for entry of typed text into a text box
`"radio"`	for the display of a radio button, usually, more than one is required
`"checkbox"`	for display of checkbox(es)
`"submit"`	for display of a submit button
`"reset"`	for display of a reset button
`"button"`	for use with JavaScript

The NAME attribute

The NAME attribute's value, chosen (invented) by you, helps to identify what data the user has typed when the e-mail message is returned to the author. (If the data is sent to a program, then the name attribute's value becomes a *variable* name. If this sounds geek to you, then feel free to

ignore it. It will be meaningful only to programmers.) We shall examine several examples below and it will then make sense.

Here is the code which creates the form in Figure 9.1. We shall dissect it now in detail.

```
<HEAD><TITLE>Registration for JavaScript
Course</TITLE></HEAD>
<BODY>
<CENTER>
<B><FONT SIZE=+1>Register for JavaScript
Course</FONT></B>
</CENTER>
<P>
<STRONG>Please enter the following details:
</STRONG>
<FORM   METHOD=POST
  ACTION="mailto:j.shelley@ic.ac.uk">
Please enter your Firstname:
<INPUT TYPE="text" NAME=" Firstname " SIZE=20>
<BR>
Please enter your Surname: 
<INPUT TYPE="text" NAME="Surname" SIZE=25>
<BR>
your e-mail address:    
<INPUT TYPE="text" NAME="Mailadd" SIZE=30>
<P>
<B>Previous JavaScript Experience</B>
<INPUT TYPE="radio" NAME="experience"
VALUE="None" CHECKED> None
<INPUT TYPE="radio" NAME="experience"
VALUE="Little"> Little
<INPUT TYPE="radio" NAME="experience"
VALUE="Famount"> Fair amount
<P>
<B>The following details will prove useful to
the Tutor:</B>
<BR>
<B>Have you:</B>
previous programming - <I>any language</I> -
experience
<INPUT TYPE="checkbox" NAME="PROG" VALUE="prog">
```

```
<BR>
used Windows 2000 or NT
<INPUT TYPE="checkbox" NAME="WIN2000"
VALUE="2000-nt">
used a Mac
<INPUT TYPE="checkbox" NAME="MAC" VALUE="mac">
<P>
<B>Have you any specific requirements?</B>
<BR>
<TEXTAREA NAME="comments" ROWS=6 COLS=35>Enter
specific requirements here.
</TEXTAREA>
<BR>
<INPUT TYPE="submit" VALUE="Submit Details">
<INPUT TYPE="reset">
</FORM>
</BODY>
```

Notes:

1. We have opted for the form's data to be e-mailed directly to the address given in the URL of the ACTION's value and, consequently, we need the POST value for the METHOD attribute.

```
<FORM   METHOD=POST
        ACTION="mailto:j.shelley@ic.ac.uk">
```

2. The text box

```
Please enter your Firstname:
<INPUT TYPE="text" NAME="Firstname" SIZE=20>
<BR>
Please enter your Surname: 
<INPUT TYPE="text" NAME="Surname" SIZE=25>
<BR>
your e-mail address:    
<INPUT TYPE="text" NAME="mailadd" SIZE=30>
```

We have three text boxes, one for the firstname, a second for the surname and a third for an e-mail address. Since we want the box to follow the description, we have put the description before the <INPUT> tag. Placing it after this tag would have put the text box first and the description would follow. It is a matter of personal style.

The browser automatically creates the various input elements. We simply have to supply the type of element we need.

The SIZE attribute
This simply specifies the *width* of the text box in characters. If the name typed in exceeds 20 characters, as specified by the `Firstname` text box, the extra characters will scroll horizontally in the box and *will be* included in the e-mail message.

The NAME attribute
Let us suppose that you have to prepare badge names for those attending the course. You want the first name followed by the surname: John Doe. How would you like it if the e-mail message sent back to your office machine simply gave the following two words: *Francis George*? Which is the firstname and which is the surname? (This implies that you have forgotten the placement of the boxes on the form. Very easy if you have to deal with registrations for several courses at the same time, each one designed, perhaps, by different people.) *George Michael*, *Shelley Francis* are other obvious examples.

The NAME attribute solves the problem. What your e-mail message will display is the name attribute's value followed by an equal symbol followed by the data typed in, thus:

```
Firstname=Francis&Surname=George
```

Now, you cannot make a mistake! Each input element is separated from the next by an ampersand (&).

(A program would use the NAME value to store the data in the correct field of the Course database. It would also use the ampersand as a separator for each piece of data.)

N.B: the use of the <BR> tag when you want to force an input element onto a new line.

3. The radio button

We need to discover what experience of JavaScript, if any, people have. Radio buttons are ideal for this. The following would display three radio buttons (little round circles which can be clicked).

```
<B>Previous JavaScript Experience</B>
<INPUT TYPE="radio" NAME="experience"
       VALUE="None" CHECKED> None
<INPUT TYPE="radio" NAME="experience"
       VALUE="Little"> Little
<INPUT TYPE="radio" NAME="experience"
       VALUE="Famount"> Fair amount
```

Why is the NAME value the same in each case - *experience*? It tells the browser that the buttons with the same NAME value are a single group of mutually exclusive buttons and that only one is allowed to be clicked. If a reader clicks one and then decides to click another, the first one will automatically be deselected by the browser. Other examples are: *sex*: Male-Female; *rating*: Good-Average-Poor. Only one can be selected. The browser is geared up to allow only one radio button to be clicked at one time and to insert a black dot in the one selected.

The VALUE attribute

The purpose of the VALUE attribute is to attach something to the NAME's value in the e-mail message. If you think about it, the text element can contain any data (any surname) but the radio button is chosen based on what the description next to it says. But this description is just ordinary text entered by the author. It cannot be 'picked up'. So, *we* have to supply what will be attached to the radio button's NAME value. Thus, in the example above, the e-mail message will contain one, and only one, of the following:

```
experience=None
experience=Little
experience=Famount
```

Note how the descriptive text is for the reader, but the VALUE text is for the e-mail message, thus Fair Amount has been shortened to `Famount`. It is the author who chooses what he/she wants to see in the actual e-mail message and what should be displayed for the reader.

The CHECKED attribute

This attribute simply puts a black dot on the radio button which has this attribute. We have put it on the first one. Note that it does not take a value. (Not all attributes need to take a value.)

My own personal opinion, take it or leave it, is never to add the CHECKED attribute, simply because some readers will forget to make a correct selection for this part and when they click on the submit button, the checked *one will end up in your e-mail message as though the reader had deliberately chosen it.*

4. The Checkbox button

```
<B>Have you:</B>
previous programming - <I>any language</I> -
experience
<INPUT TYPE="checkbox" NAME="PROG"
       VALUE="progexp">
<BR>
used Windows 2000 or NT
<INPUT TYPE="checkbox" NAME="WIN2000"
       VALUE="2000-nt">
used a Mac
<INPUT TYPE="checkbox" NAME="MAC"
       VALUE="mac">
```

The same attributes and their usage applies to checkboxes, except that the NAME-value in each case must be different since *none*, *some* or *all* of the checkboxes may be selected. Those which are not selected do not appear in the e-mail message, just those which have been selected.

5. The <TEXTAREA> tag

```
<TEXTAREA NAME="comment" ROWS=6 COLS=35>
    Please enter any comments here.</TEXTAREA>
```

There is another text input field which is used when *multiple* lines of input are requested, such as comments. This is done via the <TEXTAREA> tag, which unlike the <INPUT> tag, requires a start and an end tag. Any text placed within the pair will appear in the text area box.

We would like to have four rows (height) and 35 columns (width) to be displayed. The reader can type whatever they wish to into this area. If the 'essay' exceeds the dimensions above, scroll bars will become active. The reader is not limited to 4 lines of 35 characters each in length. That is just the dimensions the author would like the text area box to have on the web page.

(My personal style again! I prefer not to enter text between the pair of <TEXTAREA> tags, since it requires action by the user to delete that text otherwise it will be included in the e-mail message along with whatever they did type in.)

6. The SUBMIT & RESET buttons

```
<INPUT TYPE="submit" VALUE="Send Information">
<INPUT TYPE="reset">
```

In the above, two buttons would be displayed. Clicking the submit button would send off the information to the URL defined in the <FORM> ACTION attribute. Browsers are programmed to take action whenever a submit button is clicked. They look at the FORM's ACTION value to see which URL to submit the data to by the METHOD value specified.

The RESET button simply clears all the data which readers have entered so that they can start all over again, implying that they realise they have made mistakes.

Both buttons may take an optional VALUE attribute. We have given one to the submit button. The value of the VALUE attribute will appear on the button, not in the e-mail message. In our case: *'Send Information'* will appear on the submit button. The Reset button has not been given a

VALUE and some default text would appear instead, such as *Reset*. *'Send Query'* or *'Submit'* are typical default texts for the submit button when no VALUE has been given.

The e-mail message

Here is what would be sent to the author given the entries shown in Figure 9.2.

```
Firstname=John&Surname=Smith&Mailadd=jsmith@abc.co.uk
&experience=Little&PROG=prog&MAC=mac
&comments=I+need+to+know+how+to+validate+forms
+using+JavaScript +%26+I+am+also%0D%0Aa+vegetarian.
```

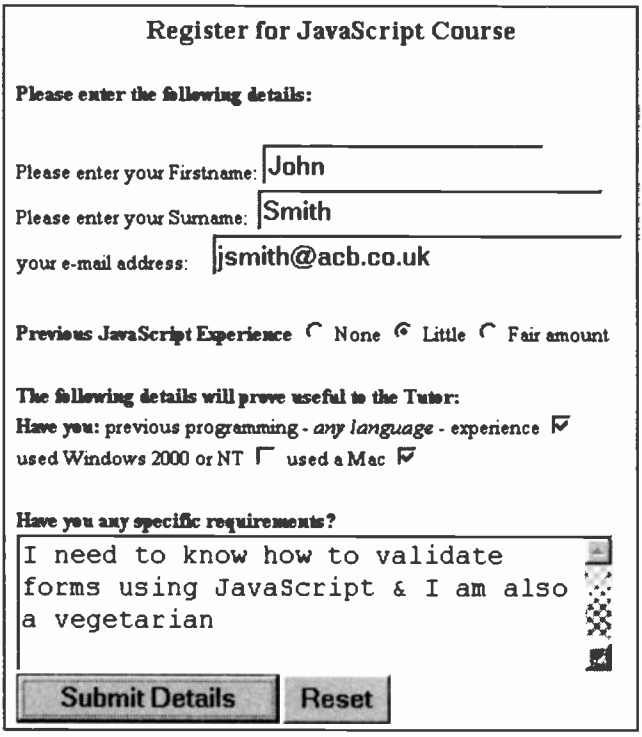


Figure 9.2

Each `NAME=value` pair is separated by `&`. Spaces are replaced with the + symbol.

The message string sent as an e-mail is difficult to read and the person receiving the message would need to convert it into a more readable format.

I tend to copy and paste it into a word processor and then use a *Find/Replace* feature to find '+' and replace it with a *nonbreaking space* character. Likewise, I find all '&'s and replace them with the *manual line break*. I also recorded a macro to automate this task. Having selected the data string, I run the macro and the entire process is automated. Here is the result which is now much easier to read:

```
Firstname=John
Surname=Smith
Mailadd=jsmith@abc.co.uk
experience=Little
PROG=prog
MAC=mac
comments=I need to know how to validate forms
using JavaScript %26 I am also%0D%0A a vegetarian.
```

But what are those strange things - `%26 %0D %0A` ?
They are ASCII characters typed in by the user in the textarea box but which cannot be printed by an e-mail program. It gives their equivalent hexadecimal number rather than the actual character. In the case of John Smith, he typed the following:

```
I need to know how to validate
forms using JavaScript & I am also (Smith pressed the Enter key)
a vegetarian.
```

%26 is the ASCII code, in hexadecimal to represent the ampersand
%0D is the *Enter key*
%0A is a *line feed*

Appendix C lists the ASCII character set where a character's hexadecimal (base 16) number is given. The %

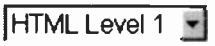
symbol is not part of the number, it is an e-mail indicator denoting an ASCII character which it cannot display.

So yes, a few strange characters will appear in your e-mail but you could always look up the ASCII character number and see what character the reader typed in.

Some e-mail programs display the form's data within the actual message box, others send the data as an *attachment*. So there will be variations between e-mail programs. You will need to find out what your particular e-mail program does.

Using Menus

In the following example, Figure 9.3, we show the use of a menu. Note the use of a drop-down arrow to reveal a list (a *menu*) of courses on offer. The reader selects one and then clicks on the submit button.



Menus are used with many web pages when it is not convenient to show all the choices via checkboxes. You have probably noticed many examples already in web pages you have looked at, such as, a choice of domain names, what country you live in, what language would you like your web page in, your type of occupation, and so on.

A menu is created with the non-empty <SELECT> tags and takes a NAME attribute.

```
<SELECT NAME="course">
       <OPTION SELECTED> HTML Level 1
       <OPTION>  HTML Level 2
       <OPTION>  CSS
       <OPTION>  DHTML
</SELECT>
```

Each item in the menu is preceded by the <OPTION> tag which is non-empty but the closing </OPTION> tag is optional. If the SELECTED attribute is used (it takes no

value), that is the one which is shown in the single menu box.

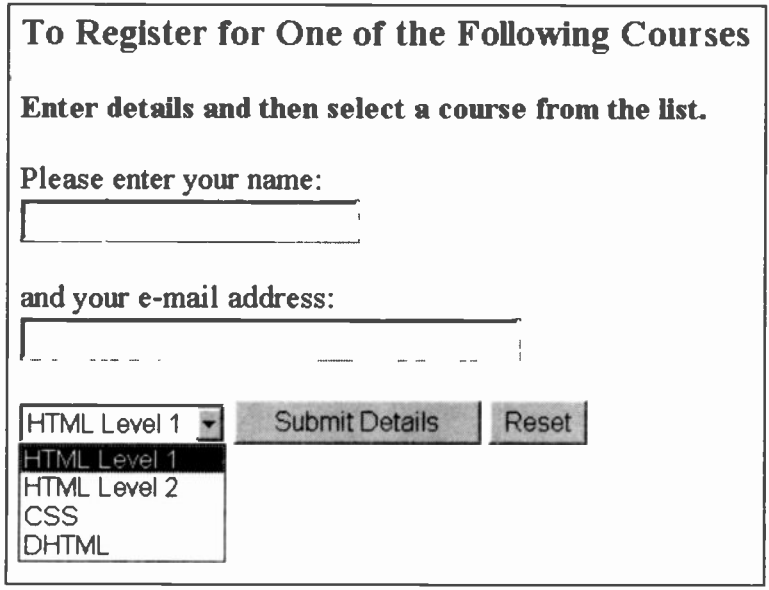


Figure 9.3

```
<HEAD>
<TITLE>Registration for Courses</TITLE>
</HEAD>
<BODY>
<H3>
To Register for One of the Following Courses
</H3>
<P>
<STRONG>
Enter details and then select a course from the
list.
</STRONG>
<P>
<FORM METHOD=POST
      ACTION="mailto:j.shelley@ic.ac.uk">
Please enter your name:<BR>
<INPUT TYPE="text" NAME="attendee" SIZE=20>
```

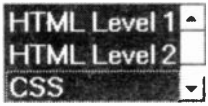
```
<P>
and your e-mail address: <BR>
<INPUT TYPE="text" NAME="mailadd" SIZE=30>
<P>
<SELECT NAME="course">
       <OPTION SELECTED> HTML Level 1
       <OPTION>  HTML Level 2
       <OPTION>  CSS
       <OPTION>  DHTML
</SELECT>
<INPUT TYPE="submit" VALUE="Submit Details">
<INPUT TYPE="reset">
</FORM>
</BODY>
```

When the drop-down list arrow is clicked, the list of courses is displayed. Note how the <SELECT> tag's NAME value ("*course*" above) will apply only to one of the options selected by the user. Like radio buttons they are mutually exclusive. The one selected will appear in the e-mail message: `course=CSS`

Using the MULTIPLE attribute

There is a MULTIPLE attribute (it takes no value) which if present allows the user to select multiple options rather like the checkbox buttons. When it is not present, the user may select only one option as with radio buttons.

If the SIZE attribute is present as well, see below, then this specifies the number of items to display, together with a scroll bar.




```
<SELECT  NAME="course"  MULTIPLE   SIZE=3>
```

There is one little problem when using the MULTIPLE attribute. How do we select more than one choice? We

cannot simply click a second choice because this de-selects the first choice!

On a Windows system, the trick is to make subsequent choices by holding down the Control key and clicking another choice. But what do Mac users do or, for that matter, users of Internet home telephones, mobiles, TV-desk tops, palm-tops, etc? Short of writing an essay explaining what to do for every device, I would prefer not to use multiple menus, but would resort to checkboxes. But you must decide for yourself. Come to think of it, I have never seen a multiple choice menu, though I have not specifically searched the web looking for one.

However, if you do decide to make use of it, perhaps because you are using it on an Intranet where you know that everyone is using a particular operating system, how does the data look in the e-mail message? Let us suppose that someone has opted for CSS and DHTML. Here is what will appear. Note how the same NAME value is attached to each selection: `course=CSS&course=DHTML`

CGI

The Common Gateway Interface (CGI) is the standard mechanism for communication between web servers and *gateway programs*. A *gateway program* is the program specifically written to process the data sent from a web page form.

When a web form is filled in and the user has clicked on the submit button, the browser sends the data to the server site where the processing program resides. That server will then activate the program and pass it the data. Once the data is processed by the program, it sends the results back to the server which then passes it back to the web browser and the user will receive the results, perhaps an acknowledgement of the receipt of some transaction.

Each server site has a special folder for storing these gateway programs, usually `/cgi-bin/`. As mentioned

above, there is one type of form design which bypasses the need for such programs or scripts, namely, when you want the data to be e-mailed *directly* to your own office computer.

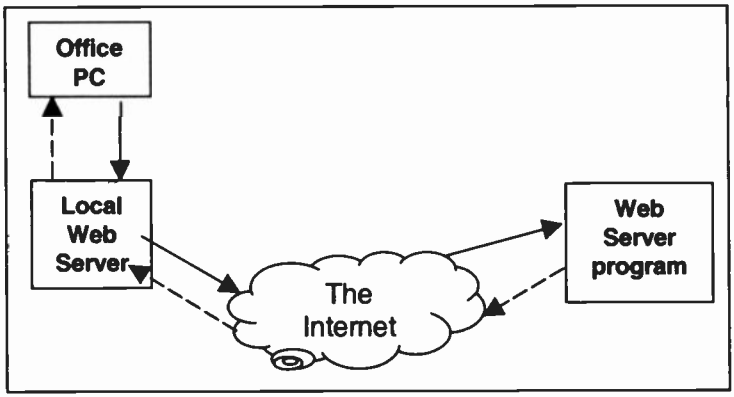


Figure 9.4
The Office browser is called the *Client*
The Web server holding the gateway program is called the *Server* site

Incorrect Form Entry

What happens when a user does not fill in the boxes correctly? The gateway program would detect the error(s) and would have to send back messages to the Client browser to request re-entry of the information. This can take time and add to the amount of information (*traffic*) passing over the Internet.

One important development in web technology is the JavaScript programming language (see Chapter 12). This language can, amongst many other things, validate data entered - or not entered - by users. The program is written by the author of the web page and included along with the HTML. When the submit button is clicked, the JavaScript program (called a *script*) is automatically activated to check all the data and will submit the form only when it is correct. If it is incorrect, the script will notify the user and ask for the correct entries to be made.

This clearly reduces the amount of traffic as well as saving the user valuable time. This is what is known as *Client-side JavaScript*,[2] where the browser via the script can complete all the error checks before submitting the data to the Server-side for processing.

What we have learnt

We have seen how to create forms for data entry and how to submit them to our own office PC. The design of forms is not a trivial exercise as many of us know when we have had to fill in forms. Some are well laid out, others can be very confusing. One of the main principles of a web form is that it should not mimic a printed version.

Frequently, we need to translate a printed form on to a web screen. If we slavishly follow the printed version, we can end up with a dreadful looking form. It is far better to sit down and revise the complete design bearing in mind that it will have to be viewed on a screen.

Summary of HTML Elements covered in Chapter 9

HTML tag	Attributes
<FORM>	METHOD = GET | POST ACTION = url *(TARGET see Chapter 11)*
<INPUT>	TYPE = text | radio | checkbox | submit | reset NAME provides a field name (not required for submit or reset buttons) VALUE (not required for text box) SIZE used with text box CHECKED radio and checkboxes only
<SELECT>	NAME MULTIPLE SIZE
<TEXTAREA>	NAME ROWS COLS

[2] See "Fun Web pages with JavaScript" by J. Shelley in this Babani series, BP520. It teaches the JavaScript language through a series of practical examples

HTML tag	Attributes
<OPTION>	used with <SELECT> for menu lists SELECTED

Test

1. Why must the value of the NAME attribute be the same in the following:

```
<INPUT TYPE="radio" NAME="gender"
       VALUE="male">Male
<INPUT TYPE="radio" NAME="gender"
       VALUE="female">Female
```

2. What are the differences between an INPUT element with TYPE="text" and the TEXTAREA text box?

3. What is the danger in using the CHECKED attribute on radio and checkbox buttons?

4. What purpose does the VALUE attribute serve in a reset button and in a checkbox button?

10: Tables & Columns

The TABLE feature allows HTML authors to do things which they could not otherwise do. For example, to create a two-column newspaper page, to put two or more images side by side on the same line, to have pretty bullet icons rather than the plain ones when using lists. It is one of the 'secrets' of design for many web pages. We shall discuss how this is done after we have studied the syntax of the <TABLE> tag.

The <TABLE> tag

Figure 10.1 shows a simple table. How is it coded in HTML? Here is the code:

```
<HEAD><TITLE>Simple Table Example</TITLE> </HEAD>
<BODY>
<H2>A Simple Table Example</H2>
<TABLE BORDER>
<TR>
<th> Header A </th> <th> Header B </th>
<th> Header C </th>
</TR>
<TR>
<td> Data 1 </td> <td> Data 2 </td>
<td> Data 3</td>
</TR>
<TR>
<td> Data 4 </td> <td> Data 5 </td>
<td> Data 6 </td>
</TR>
<TR>
<td> Data 7 </td> <td> Data 8 </td>
<td> Data 9 </td>
</TR>
</TABLE>
</BODY>
```

A Simple Table Example

Header A	Header B	Header C
Data 1	Data 2	Data 3
Data 4	Data 5	Data 6
Data 7	Data 8	Data 9

Figure 10.1

The entire table is enclosed in the non-empty pair - <TABLE> ... </TABLE> which can contain only two other tags: <CAPTION> and <TR> (for a table row). If a BORDER attribute is included in the TABLE tag, lines will be drawn around the cells.

The <TR> tag

This non-empty tag defines one row in the table and contains two other elements - <TH> (table heading) and/or <TD> (table data). The first table row tag specifies how many cells in each row. All subsequent rows *must* have the same number of cells as the first. Each row may be terminated by the optional ending tag </TR>.

The <TH> .. </TH> & <TD> .. </TD> tags

<TH> is used for cell headings. It is identical in syntax to <TD> which is used to specify what *data* appears in each cell. The only difference is that browsers usually render the heading text differently to <TD> text, typically in bold and centred. The text in a <TD> cell is not bold and is left justified.

Yes, building a table is a chore. Each row and each cell and its data has to be typed in as you can see from the corresponding code for this example. I will mention a much faster way of doing it later.

The <TH> and the <TD> tags are non-empty but the ending tags are optional since the end tag is implied by the next

<TR>, <TH> or <TD> tag. If either pair of tags contain no text then the corresponding cell is blank. The first row does not have to be a heading row with <TH> tags, unless you want that row to be bold and centred. The <TH> tags can appear anywhere within a table should you require that formatting style.

The <CAPTION> tag

Each table can have an optional CAPTION, which will centre the caption across the table and wrap text across the table if necessary, as shown in Figure 10.2. The caption can contain text, images and hyperlinks.

```
<TABLE BORDER>
<CAPTION>Some Caption for a Table</CAPTION>
<TR> ... etc.
```

Some Effects Using Tables

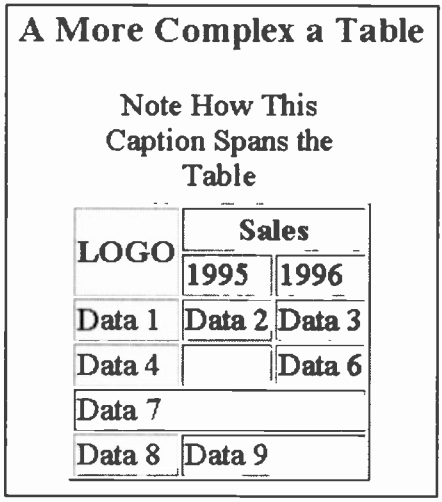


Figure 10.2

By using the ROWSPAN and COLSPAN attributes within either <TH> or <TD>, you can specify how many rows a cell can span (vertical) or columns (horizontal). For example, Figure 10.2. was achieved using the code below:

```
<BODY>
<H2>Spanning cells in a Table </H2>
<TABLE BORDER>
<CAPTION>Note How This Caption Spans the Table
</CAPTION>
<TR>
<th rowspan="2">LOGO <th colspan="2"> Sales </TR>
<TR> <td> 1995 <td> 1996 </TR>
<TR> <td> Data 1<td> Data 2 <td> Data 3 </TR>
<TR> <td> Data 4 <td>   </td> <td> Data 6
</TR>
<TR> <td colspan="3"> Data 7 </TR>
<TR> <td> Data 8 <td colspan="2">Data 9 </TR>
</TABLE>
</BODY>
```

Notes:

1. "LOGO" spans 2 rows, (in depth), but takes up 1 column (across) and is, therefore, one cell across but two cells deep.

```
<TR>
<th rowspan="2">LOGO <th colspan="2"> Sales
</TR>
```

"Sales" spans 2 columns and occupies two cells across. Therefore, the first table row specifies 3 cells in total. All other rows in the table must have the same number of cells.

2. "1995 and 1996" each occupy one column/cell. They can only go in columns 2 and 3 respectively of row 2. This is because LOGO already occupies the first column/cell of row 2 as stated in point 1 above. So there are only two cells left.

```
<TR> <td> 1995 <td> 1996 </TR>
```

3. Row 3 is normal in that it has three cells and therefore three <TD> tags.

```
<TR> <td> Data 1 <td> Data 2 <td> Data 3 </TR>
```

4. Row 4 also requires three cells but the middle cell is blank. We still need three <TD> tags albeit with nothing for

the second <TD> tag to contain. Being highly suspicious of computer programs, I always like to give a <TD> something to put in. In this case the   character entity. By doing this, you will force the browser to add a border around the middle cell, like all the other cells.

```
<TR> <td>Data 4<td>   </td> <td>Data 6</TR>
<TR> <td colspan="3"> Data 7 </TR>
<TR> <td> Data 8 <td colspan="2">Data 9 </TR>
```

Row 5 has *Data 7* spanning 3 columns.
Row 6 has *Data 8* in the first cell and *Data 9* spanning the last two columns.

If you understand the above, you have mastered tables.

Why use Tables?

1. Replacing the text *LOGO* with an <IMG> tag would have placed an image in the first two rows of column 1. Inserted between <A> tags it would have become a hyperlink image.

```
<TR> <th rowspan="2">
  <A HREF="somedoc.html">
   <IMG SRC="company_logo.gif">
  </A> </th>
 <th colspan="2"> Sales </th></TR>
```

2. The following will create a 'bulleted list' with the bullet being an image.

```
<CENTER> <TABLE>
<TR>
<TD WIDTH="10%"> <IMG SRC = "mybullet.gif">
<TD WIDTH="80%">list item 1 text </TR>
<TR>
<TD WIDTH="10%"> > <IMG SRC = "mybullet.gif">
<TD WIDTH="80%"> >list item 2 text </TR>
... etc ...
</TABLE> </CENTER>
```

Notice the use of the WIDTH attribute. This specifies the percentage of the row which one cell will occupy. In the above, we have a small image (the bullet) which requires

10% of the entire row. The second cell will occupy 80% of what is left.

What about the other 10%? How big is a table anyway? You should not ask this because it is not relevant. A table will be sized according to how much text is in any given cell. In other words, it is the content in the cells, text or image size, that govern the size of a table. However, by setting a WIDTH attribute in a TD tag, an author can specify how big a cell and all the others in the same column should be.

By putting a WIDTH attribute on the *TABLE* tag, an author can specify how much of the entire screen the table will take up regardless of the amount of content. In this case, the browser will have to wrap text onto several lines within the cell if it cannot accommodate the data on one line. (Images, of course, cannot be wrapped so their physical width must be taken into account.) You should try using the WIDTH attribute on the TD and TABLE tags to verify the above. You will soon see how they work.

3. Look at the summary of attributes at the end of this chapter. You can choose to have cells with:

- different coloured text using:
  ```
  <TD>
  <FONT COLOR="FF00E5">some text</FONT>
  </TD>
  ```
- different justifications
- different background colours
- rows can be in different colours
- watermarks can be applied to the table

With a little imagination you can create some very pleasing effects with tables.

```
<TABLE BACKGROUND="watermark.gif">
<TABLE BGCOLOR="FF5566">
<TD BGCOLOR="red">Cell 1<TD BGCOLOR="pink">Cell 2
<TR BGCOLOR="gray">
```

Note the use of the BGCOLOR and BACKGROUND attributes which behave in the same way as for the BODY

tag. Note, too, the spelling of the word 'gray' ('grey' comes out green!).

Text in Columns Using Tables

Tables may be used to create columns of text as shown in Figure 10.3.

Columns in a Table

Caption Spans Table

To create two columns, or more, of text set up a table without a border. Create one row and type in the text within the first <TD> tag.

- List item 1
- List item 2

As you can see, list items can be included. Keep going until you want to move to the next column.

This is the start of your next column of text, using a second <TD> tag. You can include not only lists, but headings, etc. Indeed, most of the HTML tags in this text.

Heading in Column 2

Use the CELLPADDING attribute of the TABLE tag to create a margin between the text and its border.
The End!

Figure 10.3

```
<BODY><CENTER>
<B><FONT SIZE+1>Columns in a Table </FONT><B>
<TABLE CELLPADDING=10>
<CAPTION>Caption Spans Table</CAPTION>
<TR>
<TD VALIGN="top" > <FONT SIZE=2> To create two
columns, or more, of text set up a table without
a border. Create one row and type in the text
within the first &lt;TD&gt; tag.
<UL>
      <LI>List item 1
      <LI>List item 2
</UL>
```

```
<P> As you can see, list items can be included.
Keep going until you want to move to the next
column.
</TD>

<TD VALIGN="top" ><FONT SIZE=2>
This is the start of your next column of text,
using a second &lt;TD&gt; tag. You can include
not only lists, but headings, etc. Indeed, most
of the HTML tags in this text.

<H4>Heading in Column 2</H4>

Use the CELLPADDING attribute of the TABLE tag
to create a margin between the text and its
border.<BR>
The End!
</TD>
</TR></TABLE>
</CENTER>
```

Notes:

1. The entire table comprises one row and two cells. When the first cell is 'finished', add a second data cell, <TD>, to create the second column.

2. Notice the VALIGN attribute for *vertical alignment*. It can take any one of the following three values: `top|middle|bottom`. This is a good way to ensure that text aligns vertically within the two columns.

3. Any HTML tag may be placed within a cell: headings, lists, images, hypertext links, font tags, and so on. The only tag which cannot be used is another table tag.

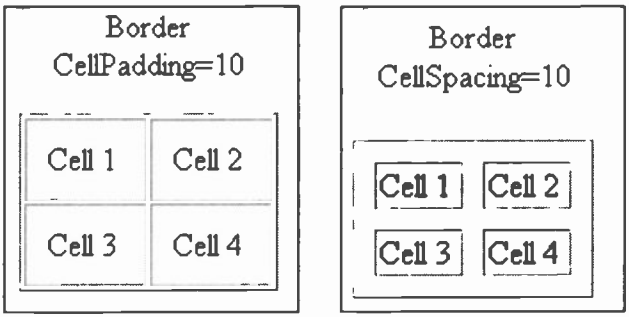
Cellpadding & Cellspacing

Cellspacing: This attribute allows for spacing between cells.

Cellpadding: This attribute allows for a padding (or margin) between text and cell borders. You may well need to experiment with these two to convince yourself of their difference.

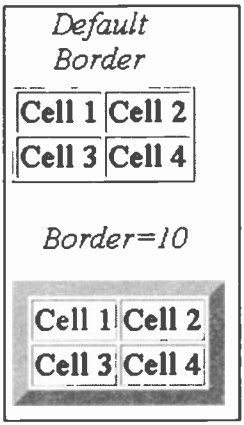
<TABLE BORDER CELLPADDING=10>

<TABLE BORDER CELLSPACING=10>



Border Thickness

<TABLE BORDER> This adds gridlines to a table.
Adding a *value* to the BORDER attribute thickens the size of a border. Thus: <TABLE BORDER=10>



Using Tables for Layout

Many web pages use tables to lay out the content rather than using them simply to present tabular data. Until cascading style sheets becomes more commonplace, page layout is achieved via tables. Here are three of the more popular 'designs' using tables. (Cascading style sheets will enable even more interesting desk top publishing features.)

1: Table Layout - 2 Columns	
link 1 link 2 link 3 ... etc. ... link n	This is really a two column table with two rows. The first row contains a Heading spanning two columns. - <TD COLSPAN = "2"> The second row contains two <TD> tags and a *width* attribute of, say, 20% in the first and 80% in the second. If the BORDER attribute of the TABLE is not present, who is to know that a table is being used?

```
<TABLE>
<TR> <TD COLSPAN="2">
<H1>1: Table Layout - 2 Columns </H1> </TD>
<TR>
<TD WIDTH= "20%" VALIGN="top">
<A HREF="url 1">link 1 </A> <BR>
<A HREF="url 2">link 2 </A> <BR>
etc... </TD>
<TD WIDTH="80%" VALIGN="top"> This is really a
two column ... etc. ... </TD>
</TABLE>
```

2: Table Layout - 3 Columns		
Links	As much text as you want can go here in the middle column of the second row.	Other related links or resources

3: Table Layout - 3 Columns again		
empty	As much text as you want can go here in the middle column of the second row. The two cells on either side of this middle cell are empty and will create white space, in effect putting in left and right margins.	empty

What you have learnt

We have seen how to create tables and discussed some of the effects you can create when using them in web pages. It is worth experimenting with tables, especially with the various attributes the table tags can take. If you see a

particularly good looking page, examine the source code and the chances are that tables have been employed to create the design.

Summary of HTML Elements covered in Chapter 10

HTML tag	Attributes
<TABLE>	WIDTH - pixel or % values BORDER BGCOLOR BACKGROUND CELLPADDING CELLSPACING
<CAPTION>	ALIGN top|bottom of table
<TR>	BGCOLOR
<TH> & <TD>	WIDTH - pixels or % values BGCOLOR COLSPAN ROWSPAN ALIGN VALIGN

Test

1. How could you place three small images side by side horizontally?

2. How could you create this effect whereby the table shows a watermark? You may need to look closely to see that the table has a watermark image!

A Watermark Example

Header A	Header B	Header C
Data 1	Data 2	Data 3
Data 4	Data 5	Data 6
Data 7	Data 8	Data 9

3. Try this in IE and then in Netscape:

```
<TABLE BORDER="5" BGCOLOR="pink"
       CELLPADDING="10" CELLSPACING="10">
```

What did you observe?

4. What is the difference between these two?

```
<TABLE WIDTH = "50%">
<TD WIDTH = "50%">
```

5. How would you get one cell coloured pink with the text in red?

6. How could you get a table to have a left border in one colour and the rest in white?

7. When using an image with the ALIGN attribute set to left or right, all following text and images will wrap around the image. How can you force just some of the text to wrap around the image with all remaining text following the image?

11: Frames in HTML

Frames permit the browser screen to be divided into multiple windows (called *frames*) with each frame containing a separate Web document. Indeed, so far, we have already been using 'frames' in the sense that our browser window has had just one frame. All that is covered will work for both Netscape and Internet Explorer.

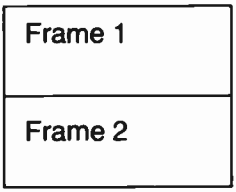
To create a web page comprising more than one frame, a separate web document is required which consists of two main tags, placed after the closing </HEAD> tag. This document has no <BODY> tag. The two tags are:

- a FRAMESET tag which specifies how many frames to create
- FRAME tags, one for each frame, to specify the content of each frame via a URL

So, if you wish to create two frames in a single browser window, *three* files would be required. A frame file (specifying the number of frames via the FRAMESET tag plus FRAME tags containing the URLs) and two more 'normal' HTML documents, one for each frame. In this text, the former will be referred to as the *frame document*.

Syntax for a Frame document

A frame document consists of one non-empty FRAMESET tag which specifies the *number* and *size* of each frame and two or more empty FRAME tags which specify the address of ordinary HTML files which each frame will contain. A frame document has no BODY tag.



A browser screen containing two web pages in rows and each 50% of the browser's window.

```
<HTML>
<HEAD>
<TITLE> Using Frames </TITLE>
</HEAD>
<FRAMESET ROWS= "50%,50%">
...<FRAME SRC="some url">
...<FRAME SRC="some url">
</FRAMESET>
</HTML>
```

a basic frame document specifying two horizontal frames, each occupying 50% of the browser's window

The FRAMESET tag

It takes one of the following two attributes: ROWS or COLS depending on whether you want the frames to be horizontal or vertical. The number of *values* after the ROW/COL attribute specify the ***number*** of rows (or columns) and each *value* specifies the ***size*** of each row (or column). Thus, the following code would create three separate *horizontal* frames (rows):

Frame A - 25% high
Frame B - 50% high
Frame C - 25% high

<FRAMESET ROWS= "25%, 50%, 25%">

The first will take up 25% of the total window, the second 50% and the last 25%. Note how the combined total percentage adds up to 100% and that each value is separated by a comma and the whole enclosed in double quotes. The following code would create three vertical frames (columns).

```
<FRAMESET COLS="20%, 50%,30%">
```

Frame A 20% wide	Frame B 50% wide	Frame C 30% wide

The ROWS Attribute

The ROWS attribute specifies horizontal frames and the *value* specifies its *height* not its width since that is determined by the user.

ROWS takes one or more of three types of *values*:

- a simple number referring to a number of pixels[1] - e.g. 100
- a percentage of the remaining space left over once the pixel value (if present) has been satisfied - a number with the % symbol, e.g. 35%
- an asterisk which takes up all the space left over after the other two types have been evaluated

Thus: `<FRAMESET ROWS="100,*,35%">` would establish a top horizontal frame of 100 pixels high, a bottom frame which takes up 35% of the *remaining* space and a middle frame which would fill up whatever space is left over.

Pixel Values

When a pixel value is used, the row height will *always* be *fixed* at that height regardless of the window size a user has established. This is the least controllable type of value to use unless other types are also included. Pixels values are always specified by a simple number.

Percentage Values

These are numbers followed by the % symbol and are limited to numbers from 1% to 100%. A row defined by a percentage value will have a *height* which fills *n%* of the window not used by other rows defined with pixel values. If all values in a row have percentages, each will take up *n%* of the window. Frames defined by % values are variable, depending on the size of the user's browser window.

Assume a row contains only percentage values and that the total values do not add up to 100%, each value is proportionally increased or decreased by the browser to make up 100%.

---

[1] Pixel is short for "picture element", i.e. all the little screen squares which make up a complete picture. 96 pixels is *about* 1 inch square on a 1024×768 resolution screen.

The Asterisk wildcard value

This is sometimes called a *wildcard variable* and consists of an asterisk. The frame it refers to is assigned whatever space is left over after pixel and/or % values have been evaluated. However, should there not be enough room left over then the wildcard row may not appear (in some browsers it may be reduced to 2 or 3 pixels). For example:

`<FRAMESET ROWS="100,80%,*">` would not leave much room, if any, for the third wildcard row.

If there are multiple wildcard variables, then the remaining space is allocated evenly between them. When a number precedes the asterisk, for example, 2*, that frame gets more space. Thus: `"2*,*"` would give 2/3rds of the space to the first and 1/3rd to the second.

Tip: *It is simpler and safer to use % values until you gain more experience in the use of frames.*

The COLS Attribute

Identical to the ROWS attribute except that since vertical frames are being described their *values* refer to the *width* of the frames, their heights being controlled by the web page reader.

It is not common to include both the ROWS and the COLS attributes in the same FRAMESET tag. It can be done but beware!

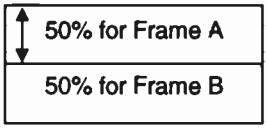
The FRAME tag

FRAME tags are placed after the FRAMESET tag. There must be one for each frame defined in the FRAMESET tag. FRAME is an empty tag and takes the following attributes:

```
<FRAME SRC="URL"
       NAME= "somename"
       MARGINWIDTH= "10" <!-- pixel values -->
       MARGINHEIGHT= "5"
       SCROLLING= "yes"|"no"|"auto"
       NORESIZE>
```

A Simple Example:

```
<FRAMESET ROWS = "50%, 50%">
   <FRAME SRC  = "frameA.htm">
   <FRAME SRC  = "frameB.htm">
</FRAMESET>
```



Since there are two frames described by the ROWS attribute, there must be two HTML documents each one referenced by a FRAME tag.

SRC attribute

This attribute provides the location of the document to be loaded into a particular frame. The source may be an HTML file, a JPEG or GIF file, etc. The syntax is the same as that for the HREF and SRC attributes of the <A> and <IMG> tags respectively. In the above, then, since we are using partial references, the browser will expect both files to co-exist in the same folder as the frameset document. Therefore, there will be three separate files, the frameset document describing the frame layout and two other documents to fill each frame.

If a FRAME tag has no SRC attribute, the frame it refers to is left blank but this can look ugly when displayed on the Web. It is always sensible to have a URL even if this is a reference to an empty file with just a background image.

We shall look at the other attributes after we have completed Exercise 1.

Exercise 1

Let us assume that we have three web documents which we want to display in three vertical frames. We, therefore, need *four* files, the frame document, and our three web documents which we have named: `fileA.htm`, `fileB.htm` and `fileC.htm`. Each will be displayed in one of the three frames.

We have decided to put *fileA* in the leftmost frame, *fileB* in the middle and *fileC* in the right frame. Each frame will occupy ⅓ of the total window.

When designing a web page with frames, the author has to work out in advance which HTML document goes into each frame. In other words, the complete layout of the overall browser screen needs to be designed before the frame document can be written.

The Frame document - *framedoc.htm*

```
<HEAD>
<TITLE> "A Simple 3 Column Design" </TITLE>
</HEAD>

<FRAMESET COLS= "33%,33%,33%">
      <FRAME SRC= "fileA.htm">
      <FRAME SRC= "fileB.htm">
      <FRAME SRC= "fileC.htm">
</FRAMESET>
```

Content for Left Frame - *fileA.htm*

```
<HEAD>
<TITLE>fileA.htm for left frame </TITLE>
</HEAD>

<BODY BGCOLOR="pink">
<!-- This goes into Left Frame -->
<H1>Contents for Left</H1>
<H3>Left Frame of the simple 3-columns</H3>
<ADDRESS>John Shelley<BR>
Content of A Frame (left)</ADDRESS>
</BODY>
```

Content for Middle Frame - *fileB.htm*

```
<HEAD>
<TITLE>fileB.htm for middle frame </TITLE>
</HEAD>

<BODY BGCOLOR="teal">
<!-- This goes into Middle Frame -->
<H1>Contents for Middle</H1>
<H3>Middle Frame of the simple 3-columns</H3>

<ADDRESS>John Shelley<BR>
Content of B Frame (middle)</ADDRESS>
</BODY>
```

Content for Right Frame - *fileC.htm*

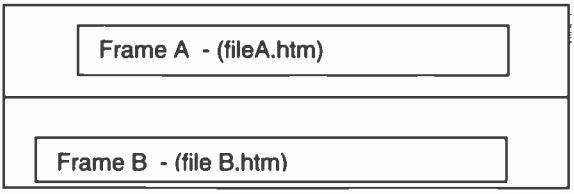
```
<HEAD>
<TITLE>fileC.htm for right frame </TITLE>
</HEAD>

<BODY BGCOLOR="lightblue">
<!-- This goes into Right Frame -->
<H1>Contents for Right</H1>
<H3>Right Frame of the simple 3-columns</H3>

<ADDRESS>John Shelley<BR>
Content of C Frame (right)</ADDRESS></BODY>
```

The MARGINWIDTH & MARGINHEIGHT attributes

These control the margins for each frame. Their values are simple numbers specifying a number of pixels between the border and the frame's content. MARGINWIDTH specifies the amount of space on the *left* margin, MARGINHEIGHT specifies the amount of space for the *top* margin. You control, therefore, only the left and top margins.

```
<FRAMESET ROWS = "50%,50%">
   <FRAME SRC = "fileA.htm"
          MARGINWIDTH= "30" MARGINHEIGHT= "15">
   <FRAME SRC = "fileB.htm"
          MARGINWIDTH= "15" MARGINHEIGHT= "30">
</FRAMESET>
```

The above code would result in two rows of equal size with the first frame's contents (`fileA.htm`) pushed 30 pixels to the right and 15 pixels down; the second frame's content (`fileB.htm`) would be pushed 15 pixels to the right and 30 down. Experiment with these to see how they work for your browser.



Notes:

1. If a frame's content extends below or to the right of the frame, scroll bars will automatically appear to allow readers to see the rest of the contents.

2. Margins settings cannot be less than 1 since contents are not allowed to touch a border. If margins are set such that none of the contents are visible, the browser will ignore your settings.

3. Both margin attributes are optional. The browser will determine appropriate settings when they are not used.

4. If a <BODY> BGCOLOR attribute is used in a frame document, that colour will fill the *entire* frame and will not be affected by any margin attribute.

SCROLLING attribute

If the frame's content cannot be contained within the frame, scroll bars will appear automatically. However, you can set the SCROLLING attribute to one of three values: "yes", "no" or "auto".

SCROLLING = "yes" means that scroll bars will *always* appear whether they are needed or not.

SCROLLING = "no" means that they will not appear, regardless of whether the contents are fully visible or not.

SCROLLING = "auto" means show the scroll bars when needed, otherwise ignore. It is the default assumed by the browser if this attribute is not used.

Some browsers may over-ride any of these settings.

NORESIZE attribute

Normally, a user can drag on a frame border to resize it. But if the NORESIZE attribute is present, it prevents the user from being able to resize the border. It takes no *value*. However, if the adjoining frame can be re-sized, this may affect a frame you have set to NORESIZE. Thus, you may need to apply this attribute to the adjoining frames as well.

Exercise 2

Here we shall create two-column frames. The web page in the left frame will contain a hyperlink to another web file. This other document will replace the existing one in the left frame and it will also contain a link back to the original.

Frame document - *Ex2_doc.htm*

```
<HEAD>
<TITLE> Two Frames with Links</TITLE>
</HEAD>
<FRAMESET COLS= "33%,33%">
  <FRAME SRC="fileA.htm">
  <FRAME SRC="fileB.htm">
</FRAMESET>
```

fileA.htm* with link to *fileC.htm

```
<BODY>
<!-- This is the original content for the left
frame -->
<H3>fileA.htm sitting in left frame</H3>
<H3>More Information below</H3>
<A HREF="fileC.htm">click here for more
information. </A>
<P> ... etc ...
<ADDRESS>John Shelley<BR>
fileA.htm in left frame </ADDRESS>
</BODY>
```

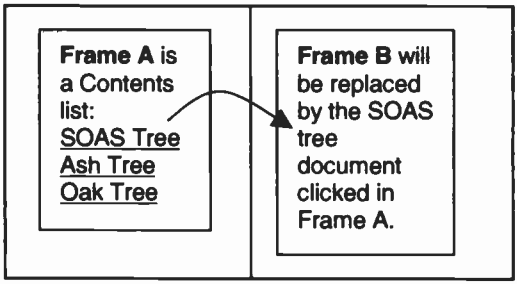
Document to replace content of left frame - *fileC.htm*

```
<BODY>
<!-- Goes into left Frame and contains a
hyperlink back to the original fileA.htm -->
<H1> New Contents for Left Frame</H1>
<H3>Link Back to Original </H3>
<A HREF="fileA.htm">Click here to
return to the original content of left FRAME</A>
<ADDRESS>John Shelley<BR>
fileC.htm replaces fileA.htm in left
frame</ADDRESS>
</BODY>
```

Target Practice

So far, we have seen how to create frames and how a hypertext link in a frame can replace its contents with another file. It is identical to a normal HTML document which replaces the full screen with another page. However, with frames, only one frame window is affected; the others are not affected.

By using a TARGET attribute, it is possible for the user to click a hypertext link in one frame and cause the HTML file to appear in *another* frame (the *target*) rather than replace the existing frame's content. This is useful when the existing frame is a list of contents which you want to remain visible all the time. This was one of the reasons for inventing frames.



In the following, we have a list of contents in frame-A. When an item is clicked, the document it links to will appear in frame-B, leaving frame-A unaltered. This is achieved via two extra attributes, a *target* attribute and a *name* attribute, thus:

Exercise 3:

The Frame Document - *Ex3_Framedoc.htm*

```
<HEAD> <TITLE> Target Practice </TITLE> </HEAD>
<FRAMESET COLS= "35%,65%">
  <FRAME SRC="contents.htm">
  <FRAME SRC= "fileF.htm"
         NAME="expendable">
</FRAMESET>
```

The NAME attribute allows the right frame to become a *target* for another file. You invent the value of this NAME but be aware that case is usually significant.

The file which is in frame-A, however, must include a TARGET attribute with the same value of "expendable" in an <A> tag.

Left frame - Contents.htm - Stays Put!

```
<HEAD> ...  etc... </HEAD>
<BODY>
<!-- This goes into Left frame as the Contents
list. -->
<H1>Contents</H1>
<H3><A HREF="SOAS.htm"
       TARGET="expendable">SOAS Tree </A>
</H3>
<H3><A HREF="ASH.htm"
       TARGET="expendable">Ash Tree</A>
</H3>
<H3><A HREF="OAK.htm"
       TARGET="expendable">Oak Tree</A>
</H3>
<ADDRESS>John Shelley<BR>
Test Case<BR>
Contents.htm</ADDRESS>
</BODY>
```

When someone clicks one of the three hyperlinks in frame-A, the browser fetches the relevant web page but will target it at the frame NAMEd *expendable*. The right frame, containing `fileF.htm` is the one NAMEd *expendable* in the frameset file called `Ex3_Framedoc.htm`.

Using TARGET with the <AREA> tag

The AREA tag may also take the TARGET attribute. In the following partial code, the first box contains the FRAMESET with the third frame being NAMEd `'exchange'`.

The first frame contains the `hawaii.htm` file. In this file, we have included a TARGET attribute within an AREA tag so that when the hot spot is clicked, `big-island.htm` will

be targeted to the third frame. In other words, `lanai.htm` will be replaced by `big-island.htm`.

```
<FRAMESET ROWS= "33%,33%,33%">
.....<FRAME SRC="hawaii.htm">
.....<FRAME SRC="maui.htm">
.....<FRAME SRC="lanai.htm" NAME="exchange">
</FRAMESET>
```

```
<!-- This is the 'hawaii.htm' file displayed
in the top row frame. -->

<MAP NAME="hawaii">
<AREA HREF="big-island.htm"
      TARGET="exchange"
      ALT="Big Island"
      SHAPE="circle"
      COORDS="394,324,74">
```

Out of interest, the FORM tag may also take the TARGET attribute. You may wish to experiment.

Reserved target names

There are four *reserved* target names, each of which begins with the underscore character. They are used as the value of the NAME attribute in the FRAME tag.

```
<FRAME SRC="hawaii.htm" NAME="_top">

<FRAME SRC="hawaii.htm" NAME="_blank">
```

Reserved names	*This causes the targeted file to be loaded:*
_blank	into a new, blank and un-named window.
_self	into the same window or frame in which the link is contained. (This is the default behaviour. It causes the existing frame to be replaced with the link document.)

Reserved names	*This causes the targeted file to be loaded:*
_parent	into the immediate FRAMESET parent which contains this target, i.e. not the document's own frameset but the next level up. If the document has no parent, then it behaves as the _self option.
_top	into the full body of the window, replacing anything which is currently there.

The `_parent` target name is used with *nested* frames, a highly complex task which requires a great deal of discipline to ensure that all frames work correctly. I have not seen many around lately, perhaps due to what follows below.

Gone are the days when everyone had to have frames and the more the merrier. Today fewer web pages contain frames for three main reasons. One is that frames take extra time to load since more than one web document has to be fetched over the Internet.

Secondly, the main reason, at least for me, is that Microsoft and Netscape went their different ways. Each invented their own attributes which would work only with their own browsers. The end result was that authors had to learn two sets of attributes and include both sets in their source code. It was all rather messy.

Thirdly, frames are *deprecated* in the new version of HTML – XHTML. The term *deprecated* means that it is a feature which is to be phased out. CSS can achieve the same effect as frames but in a more clean and simple manner.

What you have learnt

We have seen that a simple use of frames can be quite useful, especially when one contains a contents list which has hyperlinks to other documents which can be targeted for another frame, leaving the original intact on the screen.

Summary of HTML Elements covered in Chapter 11

HTML elements	Attributes
<FRAMESET>	COLS (pixels, % and * values) ROWS (pixels, % and * values)
<FRAME>	SRC NAME MARGINHEIGHT (pixels) MARGINWIDTH (pixels) SCROLLING NORESIZE

12: Where shall we be in five years?

HTML versions 2 & 3

This text has concentrated on HTML at versions 2 and 3. However, these versions are limited to the *presentation* of web pages. An author can add a splash of colour, a different typeface via the FONT tag attributes and with tables we can create certain design effects, but it is not quite desk top publishing (DTP). When a page appears, it is the same old page as the one you looked at yesterday. Yes, we can click on a hyperlink and a new web document will appear. That was the original intention of HTML. A means of exchanging text-documents between scientists resident in different countries without the need for annual Conferences. Stylish presentation was of little concern to them.

Since then the language has been hacked around to incorporate images, coloured text, different typefaces and the like. To make matters worse, both Microsoft and Netscape developed various features, mainly attributes, which the other browser was not able to display. The poor old web author was in a cleft stick. His web page would look different depending on which browser was used by the reader. A nightmare for any web author. Precautions can be taken by using a restricted set of features but the end result is a rather bland looking web page.

HTML Version 4

This version brought together two separate developments: cascading style sheets (CSS) and JavaScript.

CSS

Cascading style sheets can bring DTP to our web pages. For example, we can create line spacing, character spacing, subscripts and superscripts, small caps, drop caps, overlines; choose point sizes; create margins for a block of text without resorting to tables; create border styles

around text; position one piece of text over another; fix an image on a page so that text will scroll over it; remove underlining from hypertext, and much more. In other words our pages look more inviting.

JavaScript

The second development was JavaScript, a programming language developed by Netscape and fortunately 'adopted' by Microsoft[1], to exert control over HTML tags. This has essentially allowed readers to interact with a web page. For example, when a mouse is moved over an image, another image can take its place; images can be animated; particular images can be displayed depending on the time of day. Informative pop-up boxes can appear when a reader clicks on a button. Backgrounds colours can be changed. Goods ordered can be calculated with VAT and possible discounts and the amount displayed before the reader agrees to send off credit card details. Forms can be validated by the browser and submitted only when the form has been correctly filled in. Cookies can be created, etc.

DHTML

JavaScript can also be used to manipulate cascading style sheets to give even more control over how a web page will look. Text can be made to change colour when the mouse moves over it. Text and images can be visible or invisible when the page is loaded or at the whim of the *reader*. A short phrase can suddenly be expanded into a full paragraph as the user moves a mouse over it. An area of the current screen can be reserved for the display of different text and images when a user clicks a contents list, thus obviating the need for frames!

Combine CSS and JavaScript, and you have Dynamic HTML - DHTML. That is what version 4 of HTML is all about, web pages can become alive and vibrant. But HTML will not be developed any further. It has reached the end of the

[1] It is called JScript but is almost identical to JavaScript.

road. (Do not panic! You have not been wasting your time, as we shall see shortly.)

What is wrong with HTML?

Nothing! HTML has done a remarkably good job at popularising the growth of the WWW and all in a very short time. It was only in 1994, a bare six years ago at the time of writing, that today's 'old-hands' began to create web pages; probably just 2 - 3 years for the vast majority of Web authors.

But as with most products, HTML has come to the end of its 'best before' date. The demands of the world move on.

The new Internet

Today, most Internet access is via desktop computers. However, by 2002, it is estimated that 70% of Internet access will be via other devices: TV sets, a new breed of palm-top computers, our home telephones, talking car phones, yea, even refrigerators! HTML was not designed to cope with such diverse devices.

HTML was a language designed to mark up web documents in a *simple* way. Today's needs are more demanding. We need DTP, we want multi-media (animation, sound, music and videos in our web pages) especially with the advent of distance learning. Even `gif` and `jpeg` files leave much to be desired since they lose quality when resized. HTML cannot cope properly with such demands since it was never designed to handle them in the first place. Fudges have been attempted as is woefully obvious when viewing multi-media on many current web pages.

But of much greater significance is the fact that the Internet is fast becoming a major means for exchanging and sharing information. Information such as everyday letters, invoices, medical details, insurance details, sound, pictures, videos, book titles, newspaper, magazine and journal articles, ordering from our local supermarkets, banking,

almost anything you can think of including radio transmissions, TV material, on-line shopping. The Internet will be used more and more to transfer such data from one web site to another.

What is Needed?

HTML was defined by a special language called SGML (Standard General Mark-up Language). It is special since it is a language used to define *other* languages and how they work, for example how the syntax should be written, what type of elements (what we know as our HTML tags) should be used and for what purpose. This type of language is called a *meta-language.*

The SGML's original design for HTML stressed how to mark-up text: headings, paragraph blocks, lists, colour, size, typeface, alignment, and so on. But adding new tags to HTML is no easy task. There are technical reasons for this beyond the scope of this book as well as the fact that they would have to be sanctioned by the standards committee. Furthermore, it can take a few years for a new version to become widely used. A new web authoring language is now required but what will it be?

At the same time as a new web authoring tool is needed, there is a parallel demand for a standard way of describing data. We all know the current problems. I have a set of data in Excel but you have Lotus. Yes, you can import Excel data but you still have to thrash it around a little to meet your own needs. Transferring data between databases, spreadsheets and other programs always requires some manual effort on our part. It is not just different programs either. Try reading a Word 2000 document into an earlier version of Word.

Would it not be nice if there was one standard for describing any type of data so that any program could recognise it?

So here are the two parallel needs. A new HTML is wanted and a standard way of describing any type of data, even a web page. Enter XML! Essentially, an XML file is an ASCII file and as such can be imported into any application program, hence the ease with which its contents can be shared.

The eXtensible Mark-up Language - XML

Like SGML, XML is a special language used to define other languages. It is in fact, a sub-set of SGML but one which is easier to use and more suited to Web developers. SGML is notoriously difficult, costly and complex, reserved for use by specialists.

XML has been used to define a new web language - eXtensible HTML - XHTML. Fortunately, most of the tags used in HTML still apply to XHTML. So there is little new to learn. However, pages are formatted via style sheets rather than the <FONT> tag, which is limited anyway. An example of a style sheet is given at the end of the chapter.

So what is the big change? Web authors will be able to define their own tags at any time. It is beyond the scope of this short chapter to explain fully why this is useful but we can give an indication. XML allows users to create and define their own tags, hence the *eXtensible* in the name.

Matters get a little blurred now because we shall refer to an XML example rather than an XHTML example but there is a very close relationship between them which will have to be the subject of another book. For the moment, we shall simply try to explain why defining one's own tags can be useful.

Let us take a simple example of someone using a search engine to find the *hobbies* of a particular author, say Fred Jones. Type in his name and a search engine will throw at you every web page in its index with 'Fred Jones' somewhere in the page, even Fred Jones the builder and

Fred Jones the optician who also have their own web pages on the Internet.

Web pages using HTML can merely format text but there is no way of describing the actual *content*. But supposing we could do something like that shown on the next page.

XML (and thus XHTML) allows us to describe content by using our own tags. Imagine if you could drop the following XML code into your XHTML web page.

```
<?xml version="1.0"?>
<authorslist>
<author>
  <authorname>Fred Jones</authorname>
  <books>
    <book>
      <title>Balham - Gateway to the
             South</title>
      <isbn>142-384-8675</isbn>
      <cost>£5.99</cost>
     </book>
     <book>
      <title>My Day out in Bootle</title>
      <isbn>142-384-8765-9</isbn>
      <cost>£7.99</cost>
     </book>
   </books>
     <hobbies>
      <hobby>Acol Bridge</hobby>
      <hobby>Wimbledon: seeded number 156
             in 1997 </hobby>
     </hobbies>
    </author>
 <author>
   <authorname>Grace Armstrong </authorname>
    ...  etc.  ...
 </author>
</authorslist>
```

A search engine would now be able select all books by the author Fred Jones (or Grace Armstrong) or the hobbies, cost or ISBN number. Why? Because the elements define

what their content contains. XML, amongst other things, describes data (information). Put the above in an XHTML document and a search engine would have no difficulty in finding exactly what a user wanted. A style sheet would be used to format the content in a particular way: bold and blue in point size 14, for the *books*, italic and red and point size 12 for the *hobbies*, and so on.

> *If the above were to be opened in IE5, it would display the above almost identically. Add a style sheet, and the browser would add formatting to the contents of each tag. In other words, it would be displayed as though it had been written in HTML, just like a web page.*

This is but one simple use of XML. In the future, much of the data currently stored in our word processors, spreadsheets, databases, web pages will be converted into XML. Then any program which can recognise XML elements will be able to select any one or more of the elements and display them according to the style sheet associated with that data or use them according to some other program.

Some large organisations are already in the process of converting their existing data into XML. A mammoth task but one which will pay dividends within the next few years.

XML is not just one technology, it is the head of a family, children, cousins, aunts abound and doubtless new arrivals will appear in the future. XHTML is one of the members of the XML family and has access to the other members such as:

- SVG - scalable vector graphics
- SMIL - synchronised multi-media integrated language - pronounced 'smile'
- MathML - Mathematical markup language
- XSL - eXtensible style sheet language

SVG There are two types of graphics, bitmap and vector.

GIF, JPEG and PNG are bitmapped, that is, the files contain information about each pixel which makes up the image. A *vector* based format stores mathematical equations which describes how to redraw the image. A vector based file contains a collection of the lines, curves, and splines which are combined to draw the image. As such they are resolution independent which means that when resized they retain their original quality. In addition, the image can be made smaller or larger without significantly impacting on the size of the file.

One disadvantage of vector graphics is that they are not good at irregular images which cannot be described well by curves and lines. They are good for line art and illustrations but not for photographs.

At the present time, there is no standard for vector graphics on the Web. However, the W3C aims to make SVG the standard which will make an impact on how graphics are defined on the Web. SVG is used for describing two-dimensional graphics in XML and will support three types of graphic objects: vector-based shapes, bitmaps and text.

SMIL allows for co-ordination of multi-media elements, something which HTML cannot do.

MathML allows equations and other mathematical figures to be included in web pages more easily than with HTML.

XLS - the eXtensible Stylesheet Language - is an alternative style sheet language for styling XML documents. It is aimed at complex documentation projects. Most XHTML users will prefer to use CSS as the means for formatting their web pages.

So let us summarise the advantages of using XHTML:

- one of its main design considerations was that it should not make HTML obsolete
- it has easy access to other members of the XML family and to the advantages these can offer
- users can create their own tags

- XHTML pages can be rendered on devices other than desktop computers
- there is a broad industry support for XHTML

When HTML and XML meet, we have XHTML. It acts as a bridge between the two.

Many HTML web pages are currently being converted into XHTML so that their content can be accessed more easily in the future. Indeed, the HTML pages on your Web server may already be converted into XHTML. They will still display in exactly the same way since XHTML browsers understand HTML. For those who do not wish to convert to XHTML, they can still write their web pages in pure HTML for several years to come. However, they will be denying themselves the additional benefits of XHTML. Rather like someone who is still using WordStar or WordPerfect 4.2. (Does anyone out there remember those old word processors?)

Where do you go from here?

CSS and JavaScript are important for displaying our HTML pages in a more vibrant and colourful way, almost approaching that of DTP. That could be your next venture.

The next step is to become familiar with XHTML[1] which uses both of the above developments. Learning XHTML is quite simple and can be covered in about one hour or so on a conversion course. All tags must be in lowercase, all attributes must be double quoted. A few additional lines need to be added to the <HEAD> to inform the browser that it is an XHTML document. That is three out of the ten main differences. All the existing HTML tags still apply with the exception of the <FONT> tag since formatting is usually achieved through style sheets.

The latest version of CSS, version 3, is under development at the time of writing and will contain some very powerful formatting features.

---

[1] "XHTML and CSS explained" by John Shelley is available in this Babani series, BP501.

Finally, you may like to introduce yourself to XML. There will be a great demand for people conversant with this technology. It is in its early days as yet and there are not too many experts around.

For those who are familiar with HTML, as described in this text, converting to XHTML will not be a daunting task, so you have not been wasting your time! XHTML still uses the familiar HTML tags.

Style Sheets

Just to give a flavour of what style sheets look like and how they are used, let us suppose that we have an HTML document with five `H1` headings, five `H2` headings and ten paragraph tags.

We want headings 1 to be in red and point size 20; headings 2 to be in blue with point size 16; and, all paragraphs to be point size 13 with full justification. That would require 20 additional <FONT> tags as well as 10 <P> tag ALIGN attributes and we would *not be able* to control the point sizes.

Here is a style sheet which will do it in three lines. The style sheet is contained in a pair of `<style>` tags within the <HEAD> tags.

Curly brackets are used to enclose the styles for each of the three tags: `H1`, `H2` and `P`. Each style ends with a semi-colon.

```
<HEAD>
<TITLE> a title </TITLE>
<style>
  H1 {font-size:20pt; color:red;}
  H2 {font-size:16pt; color:blue;}
  P  {font-size:13pt; text-align:justify;}
</style>
</HEAD>
```

Now, which of the methods would you prefer to write, the one above or the one with twenty <FONT> tags? Note too how easy it would be to change the colours!

Appendix A: Miscellaneous Points

In this section we consider:

- some simple design considerations
- the design of Home Pages
- how to create web pages
- some common errors
- how to publicise your pages using TITLE & META tags

1. Some Design Considerations

HTML is quite easy to use and there is a danger that we rush into constructing our pages without due consideration to the end result. The overriding consideration is that there are many different Web browsers in use from full-blown graphical, sound and video browsers to very simple line browsers. What you see on *your* web page may look very different on *another* browser. If we remember this all the time, then we can aim to design pages which will look reasonable on any browser. It follows that you should test your pages on both Netscape and Internet Explorer, just to see what, if any, differences there may be between the two.

One of the main problems which has bedevilled web authors is that HTML is not predictable. In practice, it is nothing to do with HTML but with the browsers which interpret our HTML pages. Browsers are programs which read the source code of HTML documents. Many of them make allowances for any mistakes or non-standard use of HTML. They guess at what we may have been trying to do. This has simply led to many web authors becoming sloppy with their code and the end result is that the same page will look different depending on how tolerant one browser version is from another. One example is where I wanted a grey background colour and typed in `grey` rather than the American `gray`. This was 'interpreted', by both IE (version 5) and Netscape (4.5) as an attempt to type `green`, which was the colour I was given.

I trust that those days will soon be past as we begin to use the new web technologies such as XHTML and XML, as discussed in Chapter 12. But for the next few years we are still going to make do with HTML.

Here are a few tips which will help to make our pages more consistent across different browsers.

Keep it Simple, Sweetheart
KISS is one of the first laws of programming and applies equally to HTML. Provided you keep to the tags and their attributes as discussed in this book, then most browsers will have no problem in displaying pages along the lines *you* intend.

Using the tags mentioned in this text will ensure attractive enough documents so that you can put your effort into the *content* rather than the format of a page. After all, the reason people will want to read your pages is to get information, not pages full of images and pictures. For that they can turn to magazines. We saw in Chapter 12 that HTML was not designed to be a desk top publishing tool. If we do not get over-ambitious or until we begin to use Cascading Style Sheets, simplicity will pay its own dividends.

Dates
If you add a date, make sure that you do not use all numbers since some countries use a MM/DD/YY format whilst others use the DD/MM/YY format. 5/6/96, therefore, becomes ambiguous. Use 5/June/96 or something similar.

Dates are useful so that readers can see how up-to-date your material is. Any later refinements you make can include updating the date. However, even if you do not make any changes, you ought to keep changing the date on a regular basis. There is nothing worse than readers seeing something dated in 1996. It leaves them wondering how valid the material is. Has the author dropped dead or

moved from the organisation and no one has thought to delete or take over the document?

Signatures are often used to include the date together with a message stating whether the material is 'protected' by copyright or not.

Theft

Do not be afraid to 'borrow' other design styles which you come across and admire. Most browsers allow readers to view the HTML source of the page they are looking at. This is also a good way to pick up tips on the use of the HTML language. You can copy and paste the source into an editor program and edit the material by replacing the original with your own information. Of course, it is not allowed to use the original material in your own pages nor blatantly to adopt someone else's complete design style. But the general design may be imitated.

Avoid using *any* material where there is a strict copyright message attached, or from organisations with large funds and a virulent legal department. If you do borrow style, you should have the grace to acknowledge the fact on your own pages.

Images

As we have seen, image files are separate files which have to be collected over the Internet before they can be displayed. It is common for authors to show thumbnails of the image and to allow a reader to click a hyperlink if they want to see the full scale version. Adding the size of the file will help a reader to make that decision.

```
Click
<A HREF="large_cat.gif">
  <IMG SRC="small_cat.gif" ALT="My Cat, Charlie">
</A>
to see a larger copy of my cat (90K bytes).
```

2. Design of Home Pages

A good home page will welcome readers, let them know where they are and what interesting information they have access to. It is the starting point for the reader and should contain such details as:

- your company name and site
- who maintains the document
- what the document is about
- a list of contents which can be explored
- signature: name, e-mail, date and copyright restrictions

Many search engines will take an opening paragraph as a description of what the web page contains. Therefore, this opening paragraph should be carefully written to reflect what your page is all about.

Keep it Short and Concise

The Home page should be concise so that the reader can have it loaded quickly and then decide where to go from there. It should not contain large images which will take some time to load and make the reader despair. I have frequently given up waiting for home pages to be displayed, deciding to return some other day when I can afford the time, which probably means that I never do. If you must include graphics, then make use of thumbnails.

The information should be informative but not long-winded ramblings. The more detailed information can come from your other pages.

Clearly, the home page must have links to other documents which ideally ought to be kept in the same folder/directory as the home page so that *relative* links can be established.

Link Back to the Home Page

Once a reader has been allowed to load another page from your home page, it makes sense to have a link back to the home page. If one page leads on to another, then this could include links back to the previous and/or to the home page.

This is especially important if the reader visits one of the other pages directly without going through your home page. Links back to the home page under these circumstances will give the reader the opportunity of visiting the home page.

Likewise, if you have a long document with many *sections* which have separate links to various parts of the same document, a "Return to Top of Page" hyperlink enables the reader to get back to the top of the document quickly. Typically, the top may be a list of Contents.

Multiple Links

If your home page or any other major page has many hyperlinks, you may need to give some thought as to how they should appear. It is often confusing to readers when links are scattered within paragraphs of text. It may be better to keep them together in a list using <UL> as shown below or to use image hot spots. Which of the following would you prefer to read?

Here are some references which you may like to look at:
An Introduction to HTML is a good starting point. Whereas Further Features examines some of the HTML version 3 tags. Another useful document is The Design and Style of HTML documents. Yet more information can be found in Advanced Use of Forms.

or

Here are some references which you may like to look at:

- An Introduction to HTML is a good starting point.
- Further Features examines some of the HTML version 3 tags.
- Another useful document is The Design and Style of HTML documents.
- Yet more information can be found in Advanced Use of Forms.

Do Not Let your Visitors Escape
Some home pages list other sites and documents other than their own. This will allow the reader to click on these and be whisked off to some other site without having had the time or opportunity to explore your own information. They may never return!

3. Creating Web Documents
To create web pages, two programs are required, an editor and a browser. Any editor can be used, such as Word, Notepad, Write, etc. The browser does not need to be connected to a server. It is only used as the means by which you can see how your HTML source code will be displayed and to note any gaffes you have made.

What is important is that the source code must be saved as a *text only file* (done automatically in Notepad) but the file must have an `htm` extension. Files saved as text will automatically have a `txt` extension appended. Opening such a file in a browser will force the browser to display the *source code* not how it should look. It is the `htm` extension which forces the browser to display the source code as a web page. So remove the `txt` extension and type in `htm`. It will still be a *text only* file but will now be able to be displayed as a web page.

I personally use Word and do the following:

- *File*, *Save As*
- click the drop down arrow on *Save File As Type*
- select *Text Only*
- type in a name with an `.htm` extension and click OK

I then call up my web browser and open that file via the *File*, *Open File* option, just as if I were opening any file in Excel, PowerPoint or Word.

Whilst viewing the web page, errors, if any, will become obvious as well as any poor design features.

In Windows, both my Word and my browser remain open. They sit on the *taskbar*. Having seen an error, I simply click the Word document on the *taskbar*, make the necessary changes and click the *Save* icon. I click the browser from the taskbar and in Netscape click the *Reload* button (*Refresh* button in IE) to see what my changes now look like.

Giving you Pages to the Web Master

Once satisfied with the outcome, the web page has to be stored on a web server. This could be your organisation's server or your ISP's disc space. Many ISPs give free disc space for people to store their own personal web pages.

However, the point is that someone called a Web Master or Web Manager is in charge of where web pages are stored on their web server. The Web Master will supply you with a URL so that you can tell the world how it can locate your document. It will contain the *http* protocol, the Web *site* address, the *directory* it is stored in and the *name* of your document. In this way, and with permission from the Web Master, you can create your own home pages.

Many people add their web page address to business cards and e-mail messages. We have all seen home pages on adverts, posters, etc. That is how you tell the world about your web page.

4. Common Errors

Here are some of the common errors made even by experienced HTML practitioners.

Failure to include *both* double quotes where required. This is particularly common for HREF URLs. Some browsers will 'correct' the error, others cannot.

Failure to include the closing > for a tag or to add a space before the closing angle bracket. Since browsers tend to ignore things they do not understand, the text which follows may be formatted according to whatever tag was previously in effect.

Failure to add the closing slash (/) in an end tag or putting spaces around the slash will cause problems.

Missing out the semi-colon in a character entity (< for <) will cause problems, too.

Make sure you always have a <TITLE> tag and that your documents have only one <HEAD> and one< BODY> tag.

It is often important to close certain non-empty tags, especially the <FONT> tag. People frequently open two or more font tags but close only the last one. That means the others are still effective.

Perhaps one of the most common errors is the misuse of the <P> tag. It should not be used with those tags which imply their own structure, for example with headings, blockquote, lists, and so on.

5. Using <TITLE> & <META> tags

Automatic programs called *web crawlers, spiders, robots* - amongst other terms - trawl the Web looking for keywords in Titles, Headings and URLs, and compile databases from them. Should you want your document to be included in these databases, add a <TITLE> tag and a <META> tag within the <HEAD> of your document.

The TITLE tag should contain keywords describing the content of the document. Long titles are seldom used. To add extra keywords which can be picked up by search engines, use the META tag.

The <META> tag

The META tag is not a required tag when creating your web pages. But, if present, they can be used by search engines to list your page on their database index. You may need to register your page with one or more search engines before they will visit your site to see what your page is all about.

The META tag should be placed after the TITLE tag and usually takes the general form:

```
<META NAME="somename" CONTENT="some content">
```

It is an empty tag and should not contain any line breaks. In other words, if the content part extends over the line, just keep typing and allow word wrap to take effect.

Here are two useful META tags.

```
<META NAME = "description"
      CONTENT = "a brief description of your
                 page">
```

The content could be a word, a phrase or a paragraph. It should be kept reasonably brief.

```
<META NAME="keywords"
      CONTENT="a, list, of, keywords,
               separated, by a, comma">
```

Choose whatever keywords you think might be appropriate, each must be separated by a comma. Remember to include synonyms, Americanisms and so on. If you have a web page about cats, you could include pets, cat, kittens, even kitty. This is how I accidentally (really!) came across M/s Kitty's home page for those who wanted a spanking good time.

I have seen some keywords lasting almost a full A4 page! It deliberately included typing errors, mis-use of case and plurals. Many popular search engines are able to distinguish between these variations, so there is not the same need to go 'over the top' as in previous days.

Appendix B: Answers to Tests

Test for Chapter 2

1. *Four of the six heading tags put an automatic line space before and after the actual heading text. Which are they?*

<H1>, <H2>, <H3> and <H4>.

2. *Two of the six heading tags put an automatic line space only after the actual heading text. Which are they?*

<H5> & <H6> Many modern browsers will ignore this ruling. So, welcome to the world of Web authoring. It is not an exact science!

3. *What tag would you use to create a horizontal line across the whole width of the screen? Does this tag also have automatic spacing before and after the line?*

<HR>. Yes, it puts blank lines above and below the rule.

4. *"The <TITLE> tag puts a title on to your web page." True or false?*

False. It is used by the browser to display a title on the *browser's* blue title bar.

5. *If you accidentally put in two sets of <BODY> tags into your web page, what would a browser do?*

You cannot be sure. Some browsers will ignore the error, others may simply refuse to display anything below the second BODY tag.

6. *Which of the following tags are empty and which are non-empty tags:*
<HEAD>
 <B> <TITLE> <HR> <H4> ?

Empty:
 & <HR>. The others are non-empty.

7. *What is wrong with the following?*

```
<HEAD>The ABC plc Home Page</HEAD>
```

This is a common mistake. Many beginners think it is the heading for the web page, whereas the HEAD tag provides the browser with information about the web document.

The <HEAD> must contain a TITLE tag which is used for the title of the web page. Many browsers display a blank page if the TITLE tag is missing. The TITLE tag must be the first tag after the <HEAD>. There can be <META> tags (see page 148) and style and script tags for CSS and JavaScript code.

Test for Chapter 3

1. *Why would experienced Web designers never mix <B> and <STRONG> tags within the same sentence?*

Since some browsers may distinguish between the two tags by using different point sizes, the sentence would not look consistent. It may be that these tags will have more of a distinction in the next generation of HTML.

2. *What is a monospaced font? How does it differ from a proportional font?*

Each character, despite its shape, is given the same amount of space as any other character, hence the term mono (single) space. Proportional fonts give each character the amount of space its shape requires. Thus, the letter 'i' would be proportionally thinner than the wider letter 'w'.

3. *What tags are available for italicising text?*

`<I>`, `<EM>` and `<CITE>`.

4. *How can you view the source code of a web page?*

Almost any web page you see on your browser can have its source code made public. In all browsers, simply click on the View menu and choose *Source* (IE) or *Source Page* (Netscape). Netscape brings up its own source code page in colour codes. IE displays the source code in Notepad.

5. *Would the following heading be centred?*

```
<CENTRE> <H1>A Heading </H1> </CENTRE>
```

No. It is not the American spelling. It must be <CENTER>. All browsers would simply ignore the tag and left justify the heading text.

Test for Chapter 4

1. *What built-in structure has the <BLOCKQUOTE> tag*

It provides a line space before and after the quote and left indents the quoted text.

2. *What would be the differences when using the <DIR>, the <MENU> and the <UL> tags to create bulleted lists?*

Most browsers would not make any difference at all. The list items for the <MENU> tag may be a point size smaller.

3. *Why could the addition of comments within your source code prove useful?*

If you can come up with a good reason for using comments, then it will prove useful to *you*. Comments are used to leave messages for others who have to maintain your web pages or to remind yourself of certain actions you may need to take when revising the page.

4. *How could you get two paragraphs to be left indented?*

Put them inside a <BLOCKQUOTE> tag thus:

```
<BLOCKQUOTE>
<P> ... paragraph 1 etc. ...
<P> ... paragraph 2 etc. ....
</BLOCKQUOTE>
```

5. *Which of the following tags have built-in structure and which have none?*

```
<ADDRESS> <DIV> <BLOCKQUOTE> <UL> <MENU> <P>
```

All of them have their own built-in structure with the exception of the <ADDRESS> tag.

Test for Chapter 5

1. *What is the difference between setting the WIDTH attribute of the <HR> tag to a value of 50 or to the value of 50%?*

The value 50 is in pixels and it will be a *constant* size no matter what window size a reader is using. The 50% value is variable and asks for the rule to 50% of the window, whatever size window is used by the reader.

2. *How could you set the font for all the text in a web page to be two sizes larger than normal text with just one line of code?*

Simply add `<BASEFONT SIZE="+2">` after the BODY tag and before any text.

3. *Is anything wrong with this?* `<BODY COLOR="pink">`

A common mistake. Only the <FONT> takes the COLOR attribute. Most others, including the BODY tag require the BGCOLOR attribute.

4. *Write the code which will display the following onto a web page:*

> *"The Diego Velázquez exhibition is about ¼ of a mile along corridor D."*

```
&amp;quot;The Diego Vel&amp;#225;zquez exhibition is
about &amp;#188; of a mile along corridor
D.&amp;quot;
```

5. *Try this out:*

```
<CENTER>
<HR WIDTH="60%" COLOR="#090DEE"
    ALIGN="left" SIZE=3>
<FONT SIZE=+2
      COLOR="red"
      FACE="comic sans ms">
Please Take Notice
</FONT>
<HR ALIGN="right" WIDTH="60%"
    COLOR="#996633" SIZE=5>
</CENTER>
```

You should see something like the following:

Please Take Notice

6. *How could you indent lines of text without resorting to the blockquote tag which would indent by a fixed amount as well as create blank lines above and below the blockquote text?*

Make use of the ` ` character entity. Each one will give a space.

7. *Why is the heading "A Collection of Poems by Fred Bloggs" in yellow? The author intended it to be in normal black typeface and thought that the closing FONT tag would achieve this by turning off the red and yellow FONT colours.*

```
<H1><FONT COLOR="red">Poems on the
Underground</H1>
<FONT SIZE="2" COLOR="yellow">The London
Underground displays poems. Here is one by
Fred Bloggs:
<FONT SIZE="-2" COLOR="green"> Fred's poem
follows here ...
</FONT>
<H3>A Collection of Poems by Fred Bloggs</H3>
  ...  etc.  ...
```

This is a very common mistake in the use of the <FONT> tag and some other tags. There are three FONT tags, but only one closing FONT tag. That one will turn off the effect of the last opening FONT tag, leaving the previous two still in effect. The FONT tag `color="yellow"` is therefore still in effect. The web author must remember to close every opening tag with its closing tag.

Thus the `</FONT>` tag will close only the:

```
<FONT SIZE="-2" COLOR="green">
```

8. *How could you have small Roman numerals for your ordered lists?*

```
<OL TYPE="i">
```

The following would give:

```
<OL TYPE="A">
<LI> number one
<LI> number 2
      <OL TYPE="i">
      <LI> number 2.1
      <LI> number 2.2
      </OL>
<LI> number 3
<LI> number 4
</OL>
```

A. Item 1
B. Item 2
   i. item 2.1
   ii. item 2.2
C. Item 3
D. Item 4

Test for Chapter 6

1. *Which attributes can the anchor tag take?*

The HREF and the NAME attributes.

2. *What does this mean:*

`http://www.abc.ac.uk/crses/course.htm#courses` ?

Using the `http` protocol, call at the site `www.abc.ac.uk`, and ask for a copy of the resource held in the `crses` folder and called `course.htm` and when displayed make sure that the document is viewed at the position marked with a NAME attribute whose value is called `courses`.

3. *How many mistakes can you find in the following?*

```
<A HREF="#gothere>click to go there </A>
.............
<A NAME="#Gothere">Here I am </A>
```

There are three. First, the HREF value has no closing quote. Secondly, the NAME value must not have the hash symbol. Thirdly, the NAME value is not in the identical case to the HREF value. All three are very common mistakes!

4. *How could you get your readers to send you an e-mail message without having to leave their browsers?*

```
<A HREF="mailto:me@here.com"> send me an e-
mail</A>
```

5. *When do you have to include port numbers in a URL?*

Only when your Web Manager tells you to!

Test for Chapter 7

1. *What IMG attribute is required to display an image within a web page?*

```
SRC = "url"
```

2. *If the image you want loaded is stored at some external site what would happen if that site moved the image to some other folder without telling you?*

Tough Luck!

3. *Let us say that you see a web page with your organisation's logo on it. You would like to use it on your own web pages and have been given permission to do so. How would you obtain a copy of that logo?*

In Windows, right click on the logo image in any browser. This will display a menu. There will be an option saying: *Save Image As* (Netscape) or *Save Picture As* (IE). Choose this and when the usual *Save* dialogue box appears, enter a name or keep the existing one and choose the folder to save it in. It is now yours. Make sure you keep the original extension, either GIF or JPG.

4. *You see a picture on someone else's web site and would like to use it for your own web page. The answer to the above question tells you how simple it is to obtain that image. But could you be in breach of copyright?*

Absolutely! You must always get permission to use any image you steal off the Web if it is to be used in public.

5. *You have made an image a hyperlink, but are now dismayed to find that a horrible blue line runs around the entire image box. It looks even worse after it is clicked because now the browser puts a* purple *border around the image. How can you get rid of the border?*

Force the image up to the border by adding `BORDER=0` to the IMG tag. This prevents any border from being displayed.

6. *How can you tell whether you have Internet Assistant in your Word program?*

Click on *File*. If you see *Save as HTML*, then Internet Assistant is loaded.

Test for Chapter 9

1. *Why must the value of the NAME attribute be the same in the following:*

```
<INPUT TYPE="radio" NAME="gender">Male
<INPUT TYPE="radio" NAME="gender">Female
```

To make both radio buttons mutually exclusive. Now only one will be permitted to be selected at any given time.

2. *What are the differences between an INPUT element with TYPE="text" and the TEXTAREA text box?*

`TYPE="text"` will display a single line box. The TEXTAREA tag allows for multiple lines. Also, it is non-empty, unlike the <INPUT> tag.

3. *What is the danger in using the CHECKED attribute on radio and checkbox buttons?*

Someone may forget to change the buttons and, therefore, the incorrect values will be returned.

4. *What purpose does the VALUE attribute serve in a reset button and in a checkbox button?*

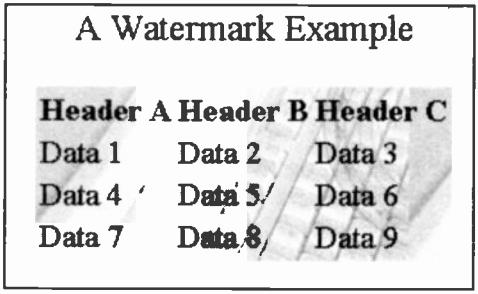
In a reset button, the *value text* will appear on the button itself. In a checkbox button, the *value text* will be sent off to the program or e-mail address.

Test for Chapter 10

1. *How could you place three small images side by side horizontally?*

Quite simply, use a TABLE with one <TR> and three <TD> tags, each one containing an image. They would have to be small enough to fit across the browser screen!

2. *How could you create this effect whereby the table shows a watermark? You may need to look closely to see that the table has a watermark image!*



In the TABLE tag, add a BACKGROUND attribute with the watermark as the image, thus:

```
<TABLE BACKGROUND="watermark.gif">
```

3. *Try this in IE and then in Netscape:*

```
<TABLE BORDER="5" BGCOLOR="pink"
       CELLPADDING="10" CELLSPACING="10">
```

What did you observe?

In Netscape, there is a border between the cells which does not let the BGCOLOR show through. In IE, the colour does show through. I prefer the Netscape effect. Which do you prefer?

4. *What is the difference between these two?*

```
<TABLE WIDTH = "50%">
<TD WIDTH = "50%">
```

The TABLE tag's WIDTH attribute controls the size of the *entire* table. It will always be 50% of whatever browser screen size is being used. The TD tag's attribute controls the size of *all cells* in its column.

5. *How would you get one cell coloured pink with the text in red?*

```
<TD BGCOLOR="pink">
<FONT COLOR="red"> The text. </FONT>
</TD>
```

6. *How could you get a table to have a left border in one colour and the rest in white?*

In the TABLE colour in the first <TD> and set an appropriate width. The rest of the table would go into the second cell.

```
<TABLE>
<TR>
<TD BGCOLOR="lightblue" WIDTH="10%">
 
</TD>
<TD>
Rest of page typed in this single cell ....
</TD>
</TABLE>
```

7. *When using an image with the ALIGN attribute set to left or right, all following text and images will wrap around the image. How can you force just some of the text to wrap around the image with all remaining text following the image?*

You simply create a single table row, with the image in one cell and the text plus any images in another cell. All subsequent text will be placed after the table. Of course, you will not use the `ALIGN=left` or `right`, since it is no longer required. Here is an image positioned on the *left* with text wrapping on the *right*.

```
<TABLE>
<TR>
<TD><IMG SRC="myimage.jpg">
<TD> the text to wrap to the right ... etc.
</TR>
</TABLE>
The rest of text to come after the image ...
```

Appendix C: The ASCII Character Set

HTML uses a basic character set ISO 8859/1, also known as Latin-1. This uses an 8-bit alphabet. A sub-set of this, ISO 646, uses a 7-bit alphabet also known as ASCII.

Computers store characters in *bytes*, a combination of 8 bits. When you type the letter 'a', this is stored internally as a unique pattern of 8 bits, 01100001, the binary equivalent of decimal 97 which is the ASCII code for letter 'a'. It is stored as a single byte. 8 bits allow for 256 (2^8) unique patterns and, therefore, 256 different characters. The first 128 characters of Latin-1 make up the ASCII character set. The remaining 128 form many of the accented and other characters used in western European languages.

The ASCII set is a 7-bit pattern, still stored as a byte, but the 8th bit is not used. Why this technical summary is important is that Macintoshes, and PCs running DOS, do not use the Latin-1 set for their internal representation of characters. Microsoft Windows and Unix on the other hand do.

This use of different character sets can and does cause problems. Fortunately, the first 128 characters of both sets are identical and make up the ASCII set. It is the second 128 characters that are different. That is why it is always safer to use the ASCII set and, when required in an HTML document, to use *character entities* (page 45) to represent any of the non-ASCII characters.

ASCII Character Set

In the following, the first 32 of the 128 characters are special and are used to control printing and communication lines. These are not printable characters and only a few are shown. No description is given when the meaning of the character is obvious. If a character has an *entity name* this is shown in the description. The name appears between an ampersand (&) and a semi-colon (;) for example, `&quot;`

Where names are not available, the ASCII code number is used preceded by a hash sign (#), thus, the tilde would be entered as a character entity as follows: ~

Note that character entity names are lowercase sensitive!

Number	Character	Description
0	NUL	Null character
7	Bell	rings a bell
8	BS	backspace
9	HT	horizontal tab
10	LF	line feed
13	CR	carriage return
32		space character - * *
33	!	exclamation mark
34	"	*"*
35	#	hash sign
36	$	
37	%	
38	&	ampersand - *&*
39	'	apostrophe
40	(	
41	)	
42	*	
43	+	
44	,	comma
45	-	hyphen
46	.	fullstop/period
47	/	solidus
48 - 57	0-9	digits 0 - 9
58	:	colon
59	;	semi-colon
60	<	*<*
61	=	equals
62	>	*>*
63	?	
64	@	commercial at
65-90	Letters A - Z	uppercase letters
91	[	

Number	Character	Description
92	\	backslash
93	]	
94	^	caret
95	_	underscore
96		acute accent
97 - 122	letters a - z	lowercase letters
123	{	left curly bracket
124	|	vertical bar
125	}	right curly bracket
126	~	tilde
127	DEL	delete

Note: There is no sterling (£) pound symbol in the ASCII character set. It is actually, number 163 in the Latin-1 set. That is why older e-mail messages had to contain the word sterling or pounds, for example:

```
Course Fee: 200.00 pounds
```

Most of our later e-mail programs can cope with many of the non-ASCII characters.

For those who are interested in the various character sets employed in computers, try this site, current at the time of writing. It is one of the most comprehensive (34 pages) articles I have found. It abounds with further references. Before I read the article I did not realise just how complex it is.

```
http://www.hut.fi/u/jkorpela/chars.html
```

For the rest of the Latin-1 character set, refer to the table on the next page. To find the decimal number for a character, use the first two digits in the left row and supply the third digit from the column headings. Thus, the yen symbol (¥) is read as 16 (row) & 5 (column), yielding the decimal number 165. Therefore, `&#165;` would give you a yen sign.

In general, HTML authors, especially those using XHTML, can safely use the ASCII characters 32 – 126, with the exception of the *less than* and *greater than* symbols (< >)

and the *ampersand* (&) since these three are regarded as mark-up characters by the browser. All the other characters from 128 – 255 should *always be given* as character entities.

Latin-1 Characters from 120 - 255

	0	1	2	3	4	5	6	7	8	9
12	x	y	z	{	|	}	~		€	
13	‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹
14	Œ		Ž			'	'	"	"	•
15	–	—	˜	™	š	›	œ		ž	Ÿ
16		¡	¢	£	¤	¥	¦	§	¨	©
17	ª	«	¬	-	®	¯	°	±	²	³
18	´	µ	¶	·	¸	¹	º	»	¼	½
19	¾	¿	À	Á	Â	Ã	Ä	Å	Æ	Ç
20	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ
21	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û
22	Ü	Ý	Þ	ß	à	á	â	ã	ä	å
23	æ	ç	è	é	ê	ë	ì	í	î	ï
24	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù
25	ú	û	ü	ý	þ	ÿ				

Note the Euro symbol (128). As a character entity this would be typed as: `&#128;` It works in IE5 but it does not work with Netscape 4.5. However, both Netscape 4.5 and IE5 accept the following character entity: `&euro;` which is easier to remember.

Most modern operating systems have the euro character installed, but if not, try the following site to download the euro symbol (current at the time of writing) by entering *euro symbol* in the *Search Box*:

`http://www.microsoft.com/`

Appendix D:

List of HTML Tags & Attributes used in this text

HTML tags	Comments & Attributes
<A> ... </A>	used for hyperlinks HREF = "url" NAME = "somename" - a place name to move to in a document
<ADDRESS> ... </ADDRESS>	renders text in italic
<AREA>	HREF ALT SHAPE = poly|circle|rect COORDS TARGET = "somename"
<B> ... </B>	bold
<BASEFONT>	SIZE sets the font size for normal text
<BIG> ... </BIG>	makes text bigger than normal text
<BLOCKQUOTE> ... </BLOCKQUOTE>	provides blank lines above and below and left indents content
<BODY> .. </BODY>	contains the content of a Web document BGCOLOR background colour for page in "#rrggbb" BACKGROUND tile an image on web page
 	force what follows onto the next line CLEAR = left|right|all It is safer to use *all*
<CAPTION> ... </CAPTION>	centres caption over a table ALIGN = top|bottom of table
<CENTER> ... </CENTER>	almost anything can be placed within this tag to centre its content on the page
<CITE> .. </CITE>	equivalent to the <I> tag

HTML tags	Comments & Attributes
<DIR> ... </DIR>	creates bulleted list
<DIV> ... </DIV>	really used with style sheets. It behaves like the <P> tag, except that it does not give an automatic line space before or after its content ALIGN = left|center|right|justify
<DL> ... </DL>	the Definition List used for creating hanging lists
<DT> ... </DT> <DD> ... </DD>	these are used with the <DL> tag for the term to be defined, <DT>, and the actual definition itself, <DD>
<EM> ... </EM>	equivalent to the <I> tag
<FONT> ... </FONT>	FACE typeface SIZE font size COLOR text colour
<FORM> ... </FORM>	METHOD = GET|POST ACTION = "url" TARGET = "somename"
<FRAME>	SRC = "url" NAME = "somename" MARGINHEIGHT (pixels) MARGINWIDTH (pixels) SCROLLING NORESIZE
<FRAMESET> ... </FRAMESET>	COLS (pixels, % and * values) ROWS (pixels, % and * values)
<H1> ... </H6>	the six heading tags ALIGN = left|center|right
<HEAD> .. </HEAD>	the head tag for browser information
<HR>	horizontal rule ALIGN = left|center|right NOSHADE (removes the groove in Netscape) - takes no value SIZE height of line in pixels WIDTH length of line, pixels or % COLOR colour of line (IE only)

HTML tags	Comments & Attributes
<HTML> .. </HTML>	defining an HTML document
<I> ... </I>	italic
<IMG>	ALIGN = left|right ALT alternative text to display for image BORDER takes a numeric value HEIGHT in pixels height of image WIDTH in pixels width of image SRC = "url" of where image is stored USEMAP use a map for hot-spots
<INPUT>	TYPE text|radio|checkbox|submit|reset NAME not required for submit or reset buttons VALUE not required for text box SIZE used with text box CHECKED for radio and checkboxes
<LI> ... </LI>	used for list items
<MAP> ... </MAP>	NAME = "somename" the value associates this MAP with a particular image via USEMAP
<MENU> .. </MENU>	alternative tag for creating bulleted lists
<META>	can be used to describe page contents NAME CONTENT
<OL> ... </OL>	ordered (numbered) list TYPE = 1|a|A|i|I
<OPTION> ... </OPTION>	used with <SELECT> to create menu list items SELECTED
<P> ... </P>	leaves a line space before text ALIGN left|center|right|justify
<PRE> ... </PRE>	the only tag which will guarantee the contained text will be in a monospaced font. White space is preserved

HTML tags	Comments & Attributes
<SELECT> ... </SELECT>	NAME MULTIPLE SIZE
<SMALL> ... </SMALL>	makes text smaller than normal text
<STRONG> ... </STRONG>	equivalent to the <B> tag
_{...}	subscripts the contained text
^{...}	superscripts the contained text
<TABLE> ... </TABLE>	WIDTH - pixel or % values BORDER takes a numeric value BGCOLOR BACKGROUND CELLPADDING takes a numeric value CELLSPACING takes a numeric value
<TEXTAREA> ... </TEXTAREA>	NAME ROWS takes a numeric value COLS takes a numeric value
<TH> ... </TH> <TD> ... </TD>	table heading (bold & centred) table data (left justified) WIDTH - pixels or % values BGCOLOR COLSPAN takes a numeric value ROWSPAN takes a numeric value ALIGN left|center|right VALIGN top|middle|bottom
<TITLE> ... </TITLE>	a mandatory title tag within <HEAD>
<TR> ... </TR>	defines a table row BGCOLOR
<TT> ... </TT> <CODE> .. </CODE> <KBD> ... </KBD> <SAMP> ...</SAMP>	all should be rendered (i.e. displayed) in a monospaced font seldom used
<U> ... </U>	the underline tag
<UL> ... </UL>	unordered (bullet) list TYPE = disc|square|circle

Glossary

ASCII	American Standard Code for Information Interchange. A code for representing characters and which is supported by almost all computer manufacturers.
attribute	in HTML, it further defines the use of a tag or element.
browsers	programs used to find and display information on the World Wide Web, WWW, using hypertext.
client	a program, such as a browser, which extracts a service or information on your behalf from a server computer somewhere on a network.
empty tag	has no associated end tag.
gateway program	a program written to process data sent from a Web form document.
HTML	the hypertext mark-up language recognised by Web browsers telling them how to display Web pages.
http	the Hyper-Text Transfer Protocol is used extensively by WWW to transfer information between networks.
hypertext	text or images which contain a link to where further information about the phrase or picture is stored. By clicking on a piece of hypertext, a new page of information pops up.
mark-up	publishing term meaning how the text and pictures are formatted.
monospace font	typeface where each character is given exactly the same amount of space.
non-empty tag	has both a start and an end tag.
platform	frequently refers to the type of operating system being used on a computer.

point size	printer's term for measuring character size. There are 72 points to an inch.
proportional font	typeface whereby each character is given the amount of space depending on its shape and width.
protocol	standards or rules which define how information is transmitted between computers.
RFC 1866	Requests For Comments 1866 which defines HTML 2.0.
server	software which allows one computer to offer a service to another computer. Client software on the other computer, requests the service from the server. Sometimes, the computer with the server software is also called the server.
SGML	the Standard General Mark-up Language which defined HTML.
tag	term for the HTML mark-up codes.
text formatter	pre-dates word processors where both the mark-up codes and the actual text had to be typed in by the author.
URL	Uniform (or Universal) Resource Locator. The method of specifying the location of resources on the Internet. Used mainly with WWW.
W3C	The World Wide Web Consortium, a body for standardising Web technologies.
Web Master or Manager	someone who maintains and mounts information on an organisation's server.
Web page	a screen display of a document found on the WWW. Confusingly, it is sometimes called a web site.
white space	printer's term referring to any form of spacing, such as tabs, normal spaces, extra spacing between words, etc.

Index

Note that 'f' stands for 'and following pages'

Babani Computer Books

How to create pages for the Web using HTML

Many companies and organisations, large, medium and small, even individuals, are now aware of the World Wide Web (WWW) as a means of publishing information over the Internet.

HTML is the current language used to create documents for Web browsers such as Netscape and Internet Explorer. These browser programs recognise this language as the method used to format the text, insert images, create hypertext, fill-in forms, coloured backgrounds and frames. HTML also forms the basis for the new genaration of Web tools, especially, XHTML and cascading style sheets (CSS).

Today, more and more people are learning HTML in order to create their own Web page documents or to amend existing pages. Fortunately, HTML is easy to learn and to use. This *Second Edition* explains the above features of the language and suggests some principles of style and design. Within a few hours, anyone could create their personal Home Page, research paper, company profile, questionnaire, etc., for world wide publication on the Web.

☑ Beginners ☑ Intermediate ☐ Advanced

BP 404

£6.99

ISBN 0-85934-[illegible]

9 780859 344043