

Contents

- 1. Introduction
 - 1.1 History
 - 1.2 XHTML
- 2. Basic Structure
 - 2.1 Elements and Attributes
 - 2.2 The HTML Document
 - 2.3 Headers and Paragraphs
 - 2.4 Some Global Attributes
 - 2.5 Adding Emphasis
 - 2.6 Layout Model
 - 2.7 Line Breaks
 - 2.8 Summary
- 3. Lists
 - 3.1 Ordered and Unordered Lists
 - 3.2 The List Element
 - 3.3 Definition Lists
- 4. Linking
 - 4.1 Anchors as Sources
 - 4.2 Uniform Resource Locators
 - 4.3 Fragment Destinations
 - 4.4 The base Element
- 5. Further Block-level Elements
 - 5.1 Introduction
 - 5.2 Preformatted Text
 - 5.3 Addresses
 - 5.4 Quotations
 - 5.5 Rules
 - 5.6 The Generic div Element
- 6. Further Inline Elements
 - 6.1 Significant Phrase Elements
 - 6.2 Mathematics
 - 6.3 Quotations
 - 6.4 The Generic span Element
- . Images
 - 7.1 Introduction
 - 7.2 The img Element
 - 7.3 The object Element
 - 7.4 Image Maps

- 8. Tables
 - 8.1 Introduction
 - 8.2 Table Structure
 - 8.3 Column Organisation
 - 8.4 Cell Organisation
- 9. Forms
 - 9.1 Introduction
 - 9.2 Form Processing
 - 9.3 The input Element
 - 9.4 Selections
 - 9.5 Writing Essays
 - 9.6 Improving Layout
 - 9.7 The button Element
 - 9.8 Summary
- 10. The Document Head
 - 10.1 Introduction
 - 10.2 The title Element
 - 10.3 The meta Element
 - 10.4 Scripting
 - 10.5 The style Element
 - 10.6 The link Element
 - 10.7 The base Element
- 11. Creating HTML
- 12. Frames

Appendices

- A. References
- B. Quick Reference Guide
- C. Deprecated Attributes

1. Introduction

- [1.1 History](#)
- [1.2 XHTML](#)

1.1 History

HTML stands for the Hypertext Markup Language. It was defined by Tim Berners-Lee in 1990 as the method of marking up pages of information to be looked at by a browser. The Standard Generalized Markup Language (SGML) had been used widely at CERN for documentation and HTML was a cut down version of the CERN documentation language. HTML is not a static language and has gone through a number of iterations over the last 10 years:

- **HTML 2.0** was the first real standard definition for core HTML features based upon current practice in 1994.
- **HTML 3.2** was W3C's first Recommendation for HTML and was defined in 1996. HTML 3.2 added tables, applets, text-flow around images, superscripts and subscripts, and was compatible with HTML 2.0.
- **HTML 4.0** was first released as a W3C Recommendation on 18 December 1997.
- **HTML 4.01** was released on 24th December 1999 and fixed a number of bugs in the HTML 4.0 specification.
- **XHTML 1.0** is a reformulation of HTML 4 in XML and was released on 26 January 2000 (updated on 1 August 2002)
- **XHTML 1.1** is a redefinition of XHTML 1.0 as a set of modules. It became a Recommendation on 31 May 2001. This Recommendation provides a consistent, forward-looking standard cleanly separated from the deprecated, legacy functionality of HTML 4.01 that was brought forward as part of XHTML 1.0.
- **XHTML Basic** was released on 19 December 2000 and is a minimal set of XHTML 1.0 functionality (defined in terms of XHTML modules) supported by a range of user agents including PDAs and mobile phones.
- **XHTML 2.0** will be the next version of HTML. It is currently at Working Draft stage. Latest Draft is 6 May 2003.

In August 2003, XHTML 1.0 (functionality defined in HTML 4.01) is the version of HTML widely supported by today's browsers. An aim from the early days of the Web was that **style** and **content** of web pages should be kept distinct. This helps to ensure that pages can be displayed on the widest set of devices and allows them to be used by people with accessibility problems (either physical or due to their location). In the early days, HTML was used for defining both the style and the content. Today, the aim is that styling should be provided through a style sheet and that HTML should be responsible for defining the content. HTML 4.01 includes a number of legacy styling attributes which are **deprecated** as such styling should be done using a style sheet. These attributes do not appear in XHTML 1.1. This Primer will stick to describing the basic features of XHTML 1.1 (functionality defined in HTML 4.01) but ignoring all the deprecated styling attributes of HTML 4.01 and XHTML 1.0.

1.2 XHTML

There is currently a move towards application-specific markup by using XML (Extensible Markup Language) as the language for defining markup languages. XML derives from SGML. This means that a reformulation of HTML as an application of XML makes a great deal of sense. The facilities available in XML and the tools surrounding XML become available. For this to happen, HTML must be written more strictly than is required in the HTML specification. For example, each start tag must have an end tag. In consequence, this Primer is described it as though all that a browser supports is XHTML 1.0. The result is a succinct Primer.

Frames in HTML 4.01 and XHTML 1.0 are not deprecated but, in many applications and environments can cause problems. In consequence there are actually six different flavours of XHTML 4.01/XHTML 1.0:

- HTML 4.01 comes in three flavours:
 - **Strict:** no styling attributes that are deprecated should appear
 - **Transitional:** styling attributes may appear
 - **Frameset:** frame functionality is also allowed
- XHTML 1.0 comes in the same three flavours:
 - **Strict:** no styling attributes that are deprecated should appear
 - **Transitional:** styling attributes may appear
 - **Frameset:** frame functionality is also allowed

2. Basic Structure

- [2.1 Elements and Attributes](#)
- [2.2 Basic Structure](#)
- [2.3 Headers and Paragraphs](#)
- [2.4 Some Global Attributes](#)
- [2.5 Adding Emphasis](#)
- [2.6 Layout Model](#)
- [2.7 Summary](#)

2.1 Elements and Attributes

Information in HTML is marked up by adding to the information **markup** that describes its structure. Marked up information consists of a set of **elements** that have the general form:

```
<xyz>Some content</xyz>
```

The terminology used is that this defines an **xyz** element. The **start tag** for the element is **<xyz>** and the **end tag** is **</xyz>**. The end tag looks very similar to the start tag except for the addition of the **/** before the name of the element. The content of the element is any text to be marked-up and can contain other elements. So, for example, this is also allowed:

```
<xyz>Some <abc>more</abc>content<def>added</def></xyz>
```

The tags must be correctly nested. This would be illegal:

```
<xyz>Some <abc>more content<def>added</abc></def></xyz>
```

Either the **def** element must be completely inside the **abc** element or completely outside it. Elements may have attributes and these form part of the start tag:

```
<xyz attr1="sometext" attr2='somemoretext'>Some content</xyz>
```

Attributes consist of the **attribute name** followed by the **equal sign** and a value which is a string of text inside one of two types of quotation marks (double quotes: ASCII decimal 34; single quotation marks: ASCII decimal 39). Two types are allowed in case the attribute's value contains quotation marks. The start and end quotation marks must either both be single or both be double quotes.

A shorthand is provided for element's that contain no content. For example:

```
<xyz attr1="sometext" attr2='somemoretext'></xyz>
```

may also be written:

```
<xyz attr1="sometext" attr2='somemoretext' />
```

2.2 The HTML Document

A strict XHTML 1.0 document would have the following structure:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
...
</head>
<body>
...
</body>
</html>
```

The document starts by indicating that it is XML and follows with a document type declaration that says what type of document this is. As we are going to be very strict in what we write, we can tell the browser this and it may help it to run more efficiently. After this comes the complete document which is an **html** element. It consists of two elements, the **head** element and the **body** element. The first contains elements that define meta information about the document while the second gives the content of the document. Most of this Primer will concentrate on defining what can be in the body of the document. Treat the first two lines as text that you have to start the document with. About the simplest document that we can have is:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
<title>My First Document</title>
</head>
<body>
<p>My first document</p>
</body>
</html>
```

The **head** consists of a single **title** element and the **body** consists of a single **p** for paragraph element. Your browser is likely to display the title as part of its decoration around the document. The document displayed will be the paragraph **My first document** displayed something like:

My first document

We have added a background to whatever our browser has produced for this page just so that the results of the markup are differentiated from the text in the Primer. We will start by looking at the elements that can be used in the **body** of the document and come back to discuss the **head** later. From now on you can assume that the document has the structure given above and all we are discussing are the elements that make up the contents of the **body** element.

The HTML document will normally be stored in a file. It is usual for that file to have an extension **.htm** or **.html**.

If we were going to be less strict and define the document as an HTML 4.01 document that allows styling attributes, the first two lines would be:

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

2.3 Headers and Paragraphs

h1 h2 h3 h4 h5 h6 p

We have already introduced the paragraph element above. HTML allows you to define 6 levels of headings with **h1** being the most important and **h6** the least important. So for example you could have:

```
<h1>1. My Book</h1>
<h2>1.1 The First Section in My Book</h2>
<h3>1.1.1 The First Sub-Section in my book</h3>
<h4>1.1.1.1 The First sub-sub-section</h4>
<p>This is the very first paragraph in my major work which tells in
great detail
all the things that I have ever done.
Sit back and enjoy it!</p>
<h4>1.1.1.2 The second sub-sub-section</h4>
<p>Which does not add a great deal to the first section which
sets the scene quite well.</p>
<h3>1.1.2 The Second Sub-Section in my book</h3>
<p>And so on. . .</p>
```

A browser might display this as follows:

1. My Book

1.1 The First Section in My Book

1.1.1 The First Sub-Section in my book

1.1.1.1 The First sub-sub-section

This is the very first paragraph in my major work which tells in great detail all the things that I have ever done. Sit back and enjoy it!

1.1.1.2 The second sub-sub-section

Which does not add a great deal to the first section which sets the scene quite well.

1.1.2 The Second Sub-Section in my book

And so on. . .

How the browser chooses to display it will depend on the browser. They are expected to make the **h1** elements more prominent than the **h2** elements and so on. Users normally do not use a higher number heading if they have not first used all the lower numbers. It would be regarded as bad practice to use **h1**, **h2** and **then h4** without using **h3** but it is not illegal. We have made the **h1** and **h2** elements more exotic by centring them and making them a different colour. Most browsers are not that adventurous and just change the font size, style and boldness to differentiate the headings.

2.4 Some Global Attributes

<h1 id= class= style= title= lang= . . . onclick= onmousedown= . . . >

There are some attributes that apply to nearly all elements. They fall into three classes. The first set identifies and styles the element more precisely, a second set concerns the language the element is written in and a third set is concerned with interacting with the element. We will only look at the first set at the moment.

The first set are:

- **id**: a unique name is assigned to the element. No other element in the document is allowed to have the same name. It is used for a variety of purposes including hypertext links, styling, interaction etc. Those uses will be discussed later or in the CSS Primer.
- **class**: at least one and possibly more class names are assigned to the element. The primary use of the class attribute is to specify the styling for elements belonging to one or more classes. The class attribute is fully described in the CSS Primer.
- **style**: styling is normally done more globally than on a single element but there are times when this may be useful. The style attribute is discussed in detail in the CSS Primer.
- **title**: the attribute adds some information about the element. For elements in the body of the document, there is no mandatory action or use for the title attribute. However it is usual for browsers to use the information in some way. For example when the cursor moves over the element, the browser may display its name as a **tool tip**.

An example of their use is:

```
<h1 title="My First Title" id="b123" class="header first topic"> . . .  
</h1>
```

Unless stated otherwise, it can be assumed that any element in the body of the document may have these attributes and these will be the only attributes unless others are mentioned. Appendix B has the element names in bold that can have all the global attributes.

2.5 Adding Emphasis

em strong

Within a heading or paragraph, it is likely that some phrase or word may need to be *emphasised* to add **strength** to some point. HTML provides two elements that can occur within the text of other elements to achieve that goal. The element **em** emphasises the enclosing text and the element **strong** emphasises it even more. It is up to the browser to decide how it will represent the two levels of emphasis. The text enclosed by the **em** element might be in italic and the text within the **strong** element might be bold but that is up to the browser. All that is hoped for is that the latter is emphasised greater than the former. An example might be:

```
<p>This is a book that you really <em>must</em> read. Phone me  
tomorrow <strong>in the afternoon</strong> and tell me what you  
think.</p>
```

This might be rendered as:

This is a book that you really *must* read. Phone me tomorrow **in the afternoon** and tell me what you think.

It might also be rendered as:

This is a book that you really *must* read. Phone me tomorrow **in the afternoon** and tell me what you think.

These are not the only elements that you can use to delineate some parts of an element like paragraph and headers. We have described these two now because they are frequently used and it allows us now to discuss a major breakdown of HTML elements into different types.

The important point in terms of the HTML document is that the reader can see that the word **must** is emphasised and that the phrase **in the afternoon** is emphasised even more. In terms of content, that is all that the reader of the page needs to know and that is all that is conveyed by the elements.

2.6 Layout Model

Some guidance is given within HTML as to how the elements in the document should appear. The headers should appear on separate lines. If there are two paragraphs, one following another then the first paragraph should normally be before the second and the end of one and the start of the other should be obvious. For emphasis, it should be clear that the emphasised words are not separate paragraphs but emphasised words within the paragraph.

In HTML, this is achieved by telling the browser what type of element each one is. They fall into two major classes:

- **inline**: elements that just contain text or other inline elements. The elements **em** and **strong** are inline elements
- **block-level**: elements that are more significant, can define their own space and can contain inline elements. The elements **p** and **h1** etc are examples of block-level elements. There are some block-level elements that are more like containers than individual block-level elements. They can contain other block-level elements. In this case the internal structure of the block is more complex. We will come across many examples of these.

Unless told otherwise, the browser is expected to present the two types differently. Block-level elements are expected to start on a newline while inline elements do not. When we get round to discuss styling, things get a bit more complicated but while we are just looking at HTML, this is sufficient in describing how the browser is expected to present the elements to the reader. The complications start to appear when there are block-level elements enclosing block-level elements. In some sense we already have that as the **body** element is a block-level element and we have shown that other block-level elements can be contained within it.

There are some elements that are used to define how other elements should appear. These are not block-level elements themselves and so fall outside the layout model. There are also elements that can really appear anywhere (script elements are a good example) but apply globally. Their appearance at a particular point in the document just to aid readability.

2.7 Line Breaks

`
`

For block-level elements like paragraphs, the normal pattern is to ignore additional white space characters like extra spaces and fill each line up before carrying on with the paragraph on the next line. If there is a need to force a line break within a paragraph, inserting the inline element **br** will cause this to happen. For example:

```
<p>This is a paragraph with a great deal  
of text in it to make sure that we have  
more than one line in it and here<br /> is where  
we want to force a line break</p>
```

This will be presented as:

This is a paragraph with a great deal of text in it to make sure that we have more than one line in it and here
is where we want to force a line break

2.8 Summary

Appendix B gives a Quick Guide to all the elements allowed in our strict version of HTML and for each it lists their attributes, whether they are inline, block or other and any special conditions that apply. Once familiar with the elements, this should be useful for jogging your memory on a particular element.

The rest of this Primer will consider the remaining elements in some detail.

3. Lists

- [3.1 Ordered and Unordered Lists](#)
- [3.2 The List Element](#)
- [3.3 Definition Lists](#)

3.1 Ordered and Unordered Lists

ul ol li

Two simple lists that are frequently used in documents are **unordered lists**, a set of bullet points that are equally important and **ordered lists** where the order of reading the list has some importance. HTML allows you to define both types of lists. The element **ul** element defines an **unordered list** and the element **ol** defines an ordered list. They both contain a set of list elements. Both use the **li** element to define the items in the list. For example:

```
<p>Here is an unordered list:</p>
<ul>
<li>A bullet point</li>
<li>Another bullet point</li>
<li>And a third</li>
</ul>
<p>Here is an ordered list:</p>
<ol>
<li>First point</li>
<li>Second point</li>
<li>Third point</li>
</ol>
```

This would appear something like:

Here is an unordered list:

- A bullet point
- Another bullet point
- And a third

Here is an ordered list:

1. First point
2. Second point
3. Third point

The only element allowed within a **ul** element or a **ol** element is a **li** element. Please note!

In this case, the browser has chosen a certain style of bullet symbol to indicate the individual items and has numbered the ordered list using the numbers 1, 2, 3 etc. Styling the document allows the user to specify different bullet symbols and different number representations (a, b, c **or** i, ii, iii even). That does not concern us here. We will just use the ones provided by the browser.

3.2 The List Element

Well it all looks pretty straightforward so far. We have a set of list elements each containing text (some of which could be emphasised) and that is as far as it goes. However, it is a great deal more complicated than that because the element `li` is a block-level element. So within an `li` element it is possible to have other block-level elements. A very obvious example is to put another list within a list, a markup that is often required. This works in much the same way for both `ul` and `ol` elements so we will only consider one of them. Let us start with our example above:

```
<p>Here is an unordered list:</p>
<ul>
<li>A bullet point</li>
<li>Point about bags and boxes and rings</li>
<li>And a third</li>
</ul>
```

Anywhere within the second bullet (or the others) we can add another ordered or unordered list so here is what happens when we insert a list after the word **bags**:

```
<p>Here is an unordered list:</p>
<ul>
<li>A bullet point</li>
<li>Point about bags
<ol>
<li>First point</li>
<li>Second point</li>
<li>Third point</li>
</ol>
and boxes and rings together</li>
<li>And a third</li>
</ul>
```

The result is as follows:

Here is an unordered list:

- A bullet point
- Point about bags
 - 1. First point
 - 2. Second point
 - 3. Third pointand boxes and rings together
- And a third

Let us go wild and put lists after the words **bags**, **boxes** and **rings**:

```
<p>Here is an unordered list:</p>
<ul>
<li>A bullet point</li>
<li>Point about bags
<ol>
<li>First bag point</li>
<li>Second bag point</li>
<li>Third bag point</li>
</ol>
and boxes
<ol>
<li>First box point</li>
<li>Second box point</li>
<li>Third box point</li>
</ol>
and rings
<ul>
<li>A ring point</li>
<li>Another ring point</li>
<li>A third point</li>
</ul>
together
</li>
<li>And a third</li>
</ul>
```

This is displayed by my browser as:

Here is an unordered list:

- A bullet point
- Point about bags
 - 1. First bag point
 - 2. Second bag point
 - 3. Third bag point
- and boxes
 - 1. First box point
 - 2. Second box point
 - 3. Third box point
- and rings
 - A ring point
 - Another ring point
 - A third point
- together
- And a third

We could go on playing this game with lists inside lists inside lists. Note that the browser has chosen a different bullet symbol for the inner list and has been kind enough to indent each sub-list to make it more readable. When we do our own styling, we can provide even more control but that is sufficient for now. However we need to show a common error that you must not do:

```
<p>The main points to be covered are:</p>
<ul>
<li>Lists</li>
<ul>
<li>Ordered Lists</li>
<li>Unordered Lists</li>
</ul>
<li>Headers</li>
</ul>
```

This looks alright and most likely the browser will accept it but **it is illegal**. Here the inner unordered list appears as an element inside the outer unordered list at the same level as the **li** elements and that is not allowed. It is only the **li** element that can have other block-level elements within it. What should be written in this case is:

```
<p>The main points to be covered are:</p>
<ul>
<li>Lists
<ul>
<li>Ordered Lists</li>
<li>Unordered Lists</li>
</ul></li>
<li>Headers</li>
</ul>
```

We have seen that we can add lists within lists but the **li** element can contain **any** block-level element so paragraphs and headers can also appear as part of the list element:

```
<p>The main points to be covered are:</p>
<ul>
<li>
<h4>Lists</h4>
<p>Ordered Lists</p>
<p>Unordered Lists</p>
</li>
<li>Headers</li>
</ul>
```

This will be displayed as:

The main points to be covered are:

- **Lists**
 - Ordered Lists
 - Unordered Lists
- Headers

3.3 Definition Lists

dl dt dd

The third type of list available in HTML is a definition list which consists of a **dl** element containing pairs of **dt** and **dd** elements. The **dt** element defines the **term** to be defined and the **dd** element gives the definition. For example:

```
<p>The following definitions apply:</p>
<dl>
<dt>Unordered List</dt>
<dd>List in any order that has a bullet in front of it</dd>
<dt>Ordered List</dt>
<dd>List in a precise order that are numbered</dd>
</dl>
```

The format for the definition list is again up to the browser. the browser might display it as:

The following definitions apply:

Unordered List

List in any order that has a bullet in front of it

Ordered List

List in a precise order that are numbered

HTML does not put any constraints on the positions of **dt** and **dd** elements within the **dl** element so that the user can define terms without giving a definition and vice versa. This can sometimes be useful when composing a document but in its final form it is sensible to have pairs of elements with each term followed by its definition. A **dt** element may be followed by several **dd** elements when there are several definitions that apply.

The **dt** element acts rather like a header element and should just contain text possibly with some inline elements to give emphasis etc. On the other hand the **dd** element is very much like the **li** element and can contain any mixture of block-level and inline elements allowed in the body of the document.

4. Linking

- [4.1 Anchors as Sources](#)
- [4.2 Uniform Resource Locators](#)
- [4.3 Fragment Destinations](#)
- [4.4 The base Element](#)

4.1 Anchors as Sources

```
<a href= >  
<h1 id= >
```

The success of the Web has come about because it allows documents on the web to link to other documents or even to parts of the current document. The **a** element, short for **anchor**, is the element that achieves that in HTML. It is mainly used to define the **source** of the link. An element within a document is defined as the **destination** of the link by the **id** attribute. For example:

```
<p>More information will be found on  
<a href="http://www.w3.org/fred/intro.htm#mydestination">the W3C site</a> .  
Please visit it.</p>
```

This will be displayed something like:

```
More information will be found on the W3C site. Please visit it.
```

Here the anchor element defines the source of the anchor by the **href** attribute. Such an anchor is displayed by most browsers with some indication that the inline text that is the content of the element is emphasised in some way. In the example above, the text is underlined, and in a different colour. Some browsers will show the source element differently depending on whether the link has been traversed before or not. Clicking on the link will cause the page with name **intro.htm** in the directory **fred** on the W3C site to be opened and positioned so the element with its **id** attribute set to **mydestination** is visible. If this is in the middle of the document, the document will be scrolled to a position where this element is visible, usually at the top of the screen. The browsers usually default to the current web page being replaced by the one referenced in the anchor element.

4.2 Uniform Resource Locators

The **href** attribute must be a **Uniform Resource Locator** although this can be relative or absolute. Given that the source document is `http://www.w3.org/fred/index.htm`, some examples of URLs are:

`http://www.w3.org/fred/intro.htm#mydestination`

A full URL that defines the **protocol** for communicating with the server (**http**), the **name** of the server (**www.w3.org**), the name of the **resource** on that server (**fred/intro.htm**) and the **element** within the resource (the one identified by **mydestination**).

`http://www.w3.org/fred/intro.htm`

As above, but the link is to the whole resource and it will normally be displayed with the start of the page visible

`http://www.w3.org/fred/`

Defines the default main document in the file **fred**. It is up to the server to define what file it returns.

`#mydest`

A relative address. It defines the element identified by the **id** attribute with value **mydest** in the same file as the source anchor. In this case it expands to `http://www.w3.org/fred/index.htm#mydest`.

intro.htm

A relative address. It defines the file **intro.htm** in the same directory as the file containing the source anchor. In this case it expands to `http://www.w3.org/fred/intro.htm`.

./intro.htm

A relative address. It defines the file **intro.htm** in the same directory as the file containing the source anchor. The dot notation moves you up a directory. In this case it expands to `http://www.w3.org/fred/intro.htm`.

../intro.htm

A relative address. It defines the file **intro.htm** in the directory above the file containing the source anchor. In this case it expands to `http://www.w3.org/intro.htm`.

../another/intro.htm

A relative address. It moves up **two** levels in the hierarchy and then the sub-directory **another**. In this case it expands to `http://www.w3.org/another/intro.htm`.

4.3 Fragment Destinations

If the destination of a link is part of a document, it is defined by the **id** attribute. For example:

```
<p>More information will be found in  
<a href="http://www.w3.org/fred/intro.htm#sec3">Section 3</a>.  
Please visit it.</p>  
. . .  
<h1 id="sec3">3. Section Three of Document<h1>
```

The **id** attribute must be unique within the document.

4.4 The base Element

```
< base href= />
```

The **base** element appears in the head of the document and there should be only one of them. It has a single attribute, **href** that defines the **base** to be used for all relative URLs in the document. In Section 4.2, the base for the examples was `http://www.w3.org/fred/index.htm`. If there was a base element in the document, the value specified by the href attribute would be used in place of this. The base element is mainly used when the same document can be retrieved from different places. The document may have arrived from one destination but the author would like all the references in the document to be made as though it came from a different origin.

5. Further Block-level Elements

- [5.1 Introduction](#)
- [5.2 Preformatted Text](#)
- [5.3 Addresses](#)
- [5.4 Quotations](#)
- [5.5 Rules](#)

5.1 Introduction

pre address blockquote hr

HTML identifies three types of paragraph for special treatment. They are:

- **Preformatted text:** there are many instances where the precise layout of a piece of text has been defined by spaces and newlines and needs to be retained.
- **Addresses:** documents normally have an author and this element is intended to convey the relevant information in order to contact that person.
- **Quotations:** borrowed material should have a link to the owner. This element is designed for that purpose.

A fourth element, the **hard rule** could either be regarded as styling or the null element. It is also described in this Section.

5.2 Preformatted Text

<pre>

The ability to output preformatted text is needed in HTML because the usual behaviour is to throw unnecessary whitespace away. For example:

```
<p>
  Cascading
  Style
  Sheets      from the
  World
  Wide
  Web
  Consortium
</p>
```

will be displayed as:

```
Cascading Style Sheets from the World Wide Web Consortium
```

This loses all the flavour of cascading that was in the original layout. Replacing **p** by **pre** as the element produces something like:

```
Cascading
Style
  Sheets   from the
    World
      Wide
        Web
          Consortium
```

The browser is expected to take account of all the whitespace and render it correctly. To help the layout of spacing to be just right, the browser is asked to use a fixed-pitch or monospaced font as has been done here. The usual method of presenting paragraphs is to fill up each line and then wrap on to the next. For preformatted text, if the line is long it should be left that way. The user will then have to scroll across to see it. One problem with preformatted text is the positioning of tabs. as no guidance is given by HTML, browsers are likely to make different decisions on what a tab means. In consequence, it is recommended not to use tabs in preformatted text.

Because preformatted text is trying to get the spacing just right, some of the inline elements are disallowed. In particular, **sup**, **sub**, **object**, **img** are not allowed to be used within a preformatted paragraph

5.3 Addresses

`<address>`

It is surprising that the **address** element has survived the various revisions of HTML. It has all the same characteristics as the **p** element, and has no additional attributes. In consequence, it could easily be achieved by having normal paragraphs with a class **address**. Browsers tend to render the element using italic script. For example:

```
<address> Bob Hopgood<br />
Oxford Brookes University<br />
bhopgood@brookes.ac.uk<br />
</address>
```

This would be rendered as:

```
Bob Hopgood
Oxford Brookes University
bhopgood@brookes.ac.uk
```

Note that the element is practically useless without forcing the line breaks by using the **br** element.

5.4 Quotations

`<blockquote cite= >`

Copyright is an important issue particularly on the Web where borrowing material is relatively easy. As the Web allows every resource to be uniquely identified, it is appropriate to define a block-level element that provides the additional attribute to point to the location from where the quotation was taken. For example:

```
<p>The role of W3C is as follows:</p>
<blockquote cite="http://www.w3.org">
<p>The World Wide Web Consortium (W3C)
develops interoperable technologies
(specifications, guidelines, software, and tools)
to lead the Web to its full potential as a forum for
information, commerce, communication, and collective understanding.</p>
</blockquote>
```

This would be displayed as:

The role of W3C is as follows:

The World Wide Web Consortium (W3C) develops interoperable technologies (specifications, guidelines, software, and tools) to lead the Web to its full potential as a forum for information, commerce, communication, and collective understanding.

Note that the browser has indented the text and changed the colour of the text as well. Unfortunately, it has not provided any mechanism for seeing the citation apart from looking at the source code.

5.5 Rules

```
<hr />
```

The **hard rule** element is probably more a styling rule than an information content one although it could be used to delineate major sections in a document. It has no additional parameters and is used as follows:

```
<p>My first paragraph</p>
<hr />
<p>My second paragraph</p>
```

This would be presented as:

My first paragraph

My second paragraph

The rule can be styled using CSS.

5.6 The Generic div Element

```
div
```

HTML has a generic block-level element called **div**. It can have any of the inline or block-level elements within it and it has no additional attributes. Its purpose is to group elements together so that they can act as a set. The element is quite powerful in terms of styling as it can have a **class** attribute defined for it so that a specific **div** element is styled differently. If the user wished to interact with a part of the document, enclosing that part with a **div** element and assigning it a unique **id** attribute ensures that the part of the document is located precisely. Until those are discussed in detail, the element does not appear to provide much additional functionality.

6. Further Inline Elements

- [6.1 Significant Phrase Elements](#)
- [6.2 Mathematics](#)
- [6.3 Quotations](#)
- [6.4 The Generic span Element](#)

6.1 Significant Phrase Elements

`cite` `dfn` `code` `samp` `var` `abbr` `acronym`

HTML provides a number of inline elements that more precisely define the semantic meaning of the enclosed text. This may be useful for the browser in terms of presentation but could also be of use to a search engine or a spell checker. They indicate phrases in the document that are not normal textual information that would appear in a paragraph. Here is an example of most of them:

```
<p>As <cite>Bob Hopgood</cite> points out, <dfn>HTML stands for
Hypertext Markup Language</dfn>, which is a <abbr title="World Wide Web
Consortium">W3C</abbr> Recommendation used by <acronym title="North
Atlantic Treaty Organisation">NATO</acronym> to markup documents.
Something like <code>class="classified"</code> is used to indicate that
some text like <samp>In Confidence</samp> is added to all the
classified paragraphs. If the <var>class</var> has a value of
<var>unclassified</var>, no additional text is added.</p>
```

This would be displayed as:

As *Bob Hopgood* points out, *HTML stands for Hypertext Markup Language*, which is a W3C Recommendation used by NATO to markup documents. Something like `class="classified"` is used to indicate that some text like `In Confidence` is added to all the classified paragraphs. If the `class` attribute has a value of `unclassified` no additional text is added.

The meaning of each is:

cite

A citation or reference to another source

dfn

This is the place where this term is defined

code

A fragment of code

samp

Sample output

var

A program argument or variable

abbr

Abbreviation, **title** attribute is used to define the abbreviation

acronym

The **title** attribute is used to define the acronym

As can be seen in the example, most browsers make some attempt at rendering the inline elements differently. Hovering over the acronyms and abbreviations should cause the meaning to be displayed as a tooltip. The element **abbr** is rarely used.

6.2 Mathematics

HTML makes no attempt to display mathematics correctly. There is a separate markup language, MathML, for that purpose. However superscripts and subscripts appear sufficiently often that markup is provided for them:

```
<p> x<sub>i</sub> = y<sub>i</sub> + 2<sup>a + b<sup>2</sup></sup></p>
```

This would be displayed as:

$$x_i = y_i + 2^{a+b^2}$$

As is shown, inline elements may have inline elements inside them if it makes sense. Broadly speaking anything that is sensible is allowed.

6.3 Quotations

```
<q cite= />
```

As well as the **blockquote** element, there is also an inline quotation element that allows quotations to occur within a paragraph. For example:

```
<p>David said, <q cite="David Duce">I saw Bob and he said <q cite="Bob Hopgood">SVG is nearly finished</q> so we had better start using it</q>.</p>
```

This would be displayed as:

David said, "I saw Bob and he said 'SVG is nearly finished' so we had better start using it".

The browser is supposed to add the quotation marks and be clever enough to use different quotation marks for the outer and inner quotes. These facilities are currently poorly supported in browsers. For example. IE6 produces no quotation marks and Opera produces the same quotation marks for both the inner and outer quotes so this is a risky element to use.

6.4 The Generic span Element

```
span
```

HTML has a generic inline element called **span**. It can have any of the inline elements within it and it has no additional attributes. Its purpose is to bracket some inline text and elements so that they can act as a set. The element is quite powerful in terms of styling as it can have a **class** attribute defined for it so that a specific **span** element is styled differently. If the user wished to interact with a part of a block-level element, enclosing that part with a **span** element and assigning it a unique **id** attribute ensures that the part of the block-level element is located precisely. Until those are discussed in detail, the element does not appear to provide much additional functionality.

7. Images

- [7.1 Introduction](#)
- [7.2 The img Element](#)
- [7.3 The object Element](#)
- [7.4 Image Maps](#)

7.1 Introduction

Inline images were added to HTML quite early on and provide a mechanism for adding pictorial information to Web pages. Two elements are provided:

img

The **img** element was defined early on and is a simple mechanism for adding images. It does not provide much control in terms of providing alternatives if either the image requested is unavailable or it is impossible for the viewer to perceive it. The aim is to supersede the **img** element with the **object** element although in August 2003 both were available for use.

object

The **object** element is similar to the **img** element but allows alternative representations so it has wider applicability. It is particularly useful when using specific image types that require additional functionality in the browser or where an alternative might be preferred due to the type of connection that the user currently has (a mobile phone, for example).

HTML does not legislate which **media types**, sometimes called **MIME type** are supported by the browser. MIME stands for Multi purpose Internet Mail Extensions. In some cases, the browser can be enhanced by adding a **plug-in** provided by a different organisation to support another MIME type. For example, Scalable Vector Graphics (SVG) pictures are currently supported in IE6 using a plug-ins provided by Adobe or Corel. If the user has not down loaded the plug-in, the SVG pictures specified cannot be displayed. If the **object** element is used to specify the SVG file to be loaded, it could provide as an alternative either the picture in a different format that most browsers do support or it could display a message telling the user to download the plug-in. The browser has to work out what type of image is being provided and in the case of the **img** element it does that by **data sniffing**, that is MIME type detection. When the server sends the file it may send the MIME type with it. The file extension will normally give a good clue to the contents. If the browser is sufficiently intelligent, it can look at the contents of the file and determine it that way. So getting the type of file requires quite a bit of work on the part of the browser and server.

In the case of the **object** element the browser retrieves the value of the **type** attribute if provided and that gives the MIME type. This means that if it does not support the MIME type, it does not need to download the file. Some of the more usual images are (filename followed in brackets by the image type):

abc.gif (image/gif)

The GIF (Graphics Interchange Format) format is widely used for bit map images that are graphics art. It is lossless, which means that all the information in the original image is kept. It suffers from some patent problems on the algorithms used for compression and so is being replaced by the PNG format

abc.png (image/png)

The PNG (Portable Network Graphics) format is similar to GIF but can handle a wider range of images with better colour support and can be more efficiently compressed.

abc.jpg abc.jpeg (image/jpeg)

JPEG (Joint Photographic Experts Group) format is widely used for transmitting real world images. It is a lossy compression which means that it approximates the original image throwing away superfluous detail.

abc.mpeg abc.mpg (application/mpeg)

The MPEG (Moving Picture Experts Group) format is for transmitting video. It provides high compression by being lossy and only transmitting the changes from one frame to the next.

abc.cgm (image/cgm)

The CGM (Computer Graphics Metafile) format is designed for transmitting line drawings. It is an ISO standard and there is a WebCGM profile for use on the Web.

abc.svg (image/svg+xml)

The SVG (Scalable Vector Graphics) format is designed as an XML application providing line drawing functionality similar to CGM but it also has better text drawing capabilities and special effects including animation. It is the standard on the web for pictures that are not bit map images.

Other possibilities that might be recognised are text/richtext, text/html, audio/wav, video/avi, application/pdf, etc. The list is very long.

7.2 The img Element

```
<img src= alt= height= width= longdesc= usemap= />
```

The inline **img** element defines an image to be displayed inline. If there is insufficient space on the line for the image, it will be placed on the next line. Better control of images is available through styling. The main attributes are:

src

Gives the URL of the file containing the image

alt

An alternative text description of the image. If the reader has the browser set **not** to display images, the **alt** text will be displayed instead. Some browsers provide the option that the text is displayed when you hover over the image. For viewers **listening** to the page via a text-to-speech browser, the **alt** text is spoken in place of the image. The **alt** attribute should always be provided.

width, height

These two attributes define the width and height of the image in pixels. If they are not provided, the browser downloads the image and works out the values for itself. This means that the rest of the page cannot be composed until the image arrives. Good practice is for these two attributes always to be provided. If their values are different from the size of the image, the browser will scale the image until it fits

longdesc

Gives the URL of a file that provides additional information about the image. For complex images where the designer wants to provide sufficient information for, say, a blind person to get an in depth description of the image, the **longdesc** attribute should be provided. When the image has an associated image map (to be discussed later), this can be used to provide additional information about the image map's contents.

usemap

For handling client side image maps, to be discussed later.

Here is an example that displays a JPEG image called **bobbin.jpg** that is 349 pixels wide and 91 pixels high:

```
<p>Hanging bobbins like this one:  
  
were sold at hangings in England as souvenirs.  
The actual size is smaller:  
  
and I can make them look silly by changing the aspect ratio.  
Here the height has been halved:  
  
and here the height has been doubled and the width halved:  
 </p>
```

This would appear as:

Hanging bobbins like this one:  were sold at

hangings in England as souvenirs. The actual size is smaller:  and I can make them look silly by changing the aspect ratio. Here the height has been halved:  and here the height has been doubled and

the width halved: 

The file **bobbin.txt** might contain:

William Bradbury was murdered near Luton. William Worsley and Levi Welch were both charged with the crime but Levi Welch turned king's evidence admitting that he had robbed him but that Worsley had struck the blow that killed him. William Worsley was found guilty of highway robbery and murder. His death was the last public hanging to take place in Bedford on 31 March 1868. Bobbins were produced as souvenirs for the men in the crowd to give to their wives as a memento of the occasion. Being the last public hanging, this bobbin was produced in a reasonable quantity and so are not as rare as other hanging bobbins. At auction in good condition they are worth around 150 pounds sterling. In a shop you might pay a significant mark up on that price.

7.3 The object Element

`<object data= type= standby= height= width= >`

The **object** element is designed to replace the **img** element and to provide facilities for inserting into a Web page files in a whole variety of formats.

The inline **object** element defines one of a set of objects to be displayed inline. The main attributes are:

data

Gives the URL of the file containing the image

type

Gives the MIME type for the object to be downloaded. The browser can check this before attempting a download.

width, height

These two attributes define the width and height of the object in pixels. If they are not provided, the browser downloads the object and works out the values for itself. Good practice is for these two attributes always to be provided.

standby

Gives a text message for the browser to display while the object is being downloaded.

Here is the previous example that displays an image called **bobbin** that is 349 pixels wide and 91 pixels high. The image is available in JPEG, GIF and PNG formats:

```
<object width="349" height="91" data="bobbin.jpg" type="image/jpeg"
standby="JPEG image downloading">
<object width="349" height="91" data="bobbin.png" type="image/png"
standby="PNG image downloading" />

No image available
</object>
```

In this case the browser will first attempt to download the JPG object. If this is not possible for any reason, it will try the PNG object and if that is also not possible, it will try to download the GIF image using the **img** element. If all three are not possible, it will display the text **No image available**. Adding an **img** element using a format that is almost certainly going to be available is useful for browsers that still do not recognise the **object** element. They will ignore those and just display the GIF image.

At the moment (August 2003), the support for the **object** element is improving but still not complete in some browsers. In consequence for most image formats (jpg, png, gif) it is often useful to continue with the **img** element.

7.4 Image Maps

```
<img usemap= />
<map name= >
<area href= alt= shape= coords= />
```

It is possible to use an image as a mechanism for linking to other pages in the same way that anchors are used. To achieve this, it is necessary to define the parts of the image that will be activated when a click is made over them and also the action to take when this happens. For example:

```
<h3>Robert Cailliau and Murray Maloney</h3>

<a href="#dummy">

</a>
</p>
<map name="maprcmm">
<area href="http://cern.web.cern.ch/CERN/Divisions/ETT/WPE/People/RobertCailliau/"
shape="rect" coords="0, 0, 138, 127" />
<area href="http://www.muzmo.com/Murray/" shape="rect" coords="139, 0, 277, 127" />
</map>
```

This would appear as:

Robert Cailliau and Murray Maloney



The **map** element can be anywhere in the file and the attribute **name** is used by the **img** element to indicate which map is being used. The **img** element has its **ismap** attribute set to the name of the **map** element. To ensure that the inline **img** element is recognised as an anchor, it is enclosed by an **a** element where the **href** attribute is not used and can be set to anything. It should be included as some browsers check that it is present.

Within the **map** element is a set of **area** elements each of which define part of the image as a clickable region and the destination of the link when a user clicks on that part of the image. In the example, there are two areas. The **coords** attribute defines the part of the image that is active and the shape of the area is defined by the **shape** attribute. The example has two rectangular areas and the four coordinates define positions on the image in pixels. The four coordinates define **minx miny maxx maxy** where the origin is the top left corner of the image and **y** increases in the downwards direction. In the example, the two areas are the left side of the photograph and the right side of the photograph. Clicking on one or other will cause a link to the appropriate home page of Robert Cailliau at CERN or Murray Maloney at his company, Muzmo Communications. The possible shapes are:

default

Defines the whole area

rect

A rectangular area where the **coords** attribute consists of **minx miny maxx maxy**

circle

A circular area where the **coords** attribute consists of **centrex centrey radius**

poly

A polygonal area where the **coords** attribute consists of **x1 y1 x2 y2 x3 y3 . . .**

There are a number of editors on the market that aid in the production of image maps. In most cases it is possible to achieve the same effect either by careful styling or by the use of Scalable Vector Graphics possible with associated images. This is probably the least useful method of associating links with images.

8. Tables

- [8.1 Introduction](#)
- [8.2 Table Structure](#)
- [8.3 Column Organisation](#)
- [8.4 Cell Organisation](#)

8.1 Introduction

HTML provides good support for tables. The table is divided into three main parts, a header, a body and a footer. There is also an optional caption that can be added to the table. Inside the three main parts, a table is divided into rows each having a number of columns. As with most word processing systems it is possible to give groups of columns special features and to merge two or more columns together or two or more rows together. The table is basically made up of cells and the content of a cell may be any set of block-level elements rather like the **li** element. So it is possible to put **another** table inside a single cell. In consequence, almost any table that is required can be achieved. CSS provides good control of the styling of individual cells and the table decoration. We shall not consider that further here.

8.2 Table structure

```
<table summary= width= >
<caption>. . .</caption>
<colgroup align= span= width= />
<col align= span= width= />

<thead>
. . .
</thead>
<tfoot>
. . .
</tfoot>

<tbody>
. . .
</tbody>

</table>
```

The table head and foot are very similar to the table body. The main difference is that the browser is supposed to output the table head and table foot on every page of the output if it is a very long table. Most of the browsers do not appear to have implemented this. Multiple table bodies are allowed and this can be useful if different parts of the table are to be styled differently. Here is a very simple example:

```

<table title="Membership of W3C July 2001" summary="5 column table with
header row and first column the type of membership. Total in the final
column and bottom row" width="90%">
<caption>W3C Membership</caption>
<thead>
<tr>
<th>Type</th><th>Americas</th><th>Europe</th><th>Pacific</th><th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<th>Full</th><td>62</td><td>29</td><td>17</td><td>108</td>
</tr>
<tr>
<th>Affiliate</th><td>240</td><td>123</td><td>47</td><td>410</td>
</tr>
<tr>
<th>Total</th><td>302</td><td>152</td><td>64</td><th>518</th>
</tr>
</tbody>
</table>

```

This would be displayed something like:

| W3C Membership | | | | |
|----------------|----------|--------|---------|-------|
| Type | Americas | Europe | Pacific | Total |
| Full | 62 | 29 | 17 | 108 |
| Affiliate | 240 | 123 | 47 | 410 |
| Total | 302 | 152 | 64 | 518 |

The sections of the table are made up of table rows denoted by the `tr` element. Each row consists of a set of entries which can either be `td` elements (for data) or `th` elements (for headers). Note that the header elements can be anywhere. In this case the table has them in the top row and first column and the overall total. The browser is expected to render the two elements differently, the browser will most likely make the header data elements (`th`) bolder than the normal data elements (`td`).

The table has been defined as having a width of 90% of the width of the page. Any of the usual length values may be used but it is quite common to specify the width as a fraction of the current window size.

The `summary` element is aimed at users who for one reason or another will have difficulty seeing the table in full. They might be blind or have a very small display or have had the table converted to text to transmit it by voice. The aim is to give some information to make it easier to understand the table. Other facilities are provided to help the user understand the table structure. Normally the summary would not be this long.

8.3 Column Organisation

The `col` and `colgroup` elements allow the author to define properties for a whole column or group of columns.

The `colgroup` element is provided to create **structural divisions** in the table. For example, the table above might have immediately after the caption:

```

<colgroup span="1" width="30%" />
<colgroup span="3" width="10%" />
<colgroup span="1" width="30%" />

```

This would change the table to look like:

| W3C Membership | | | | |
|----------------|----------|--------|---------|-------|
| Type | Americas | Europe | Pacific | Total |
| Full | 62 | 29 | 17 | 108 |
| Affiliate | 240 | 123 | 47 | 410 |
| Total | 302 | 152 | 64 | 518 |

The first column, being the headings for each row is put into one group, the three columns giving the information is a second group while the third group is the one containing the totals. The attribute **span** defines the number of columns in the group in each case. In the example, the widths have been made difference to demonstrate this. Styling can be applied differently to each of the groups.

The **col** element **shares attributes** between several columns. For example:

```
<colgroup span="3" align="right">
  <col width="30%">
  <col span="2" width="10%">
</colgroup>
<colgroup span="2" width="20%">
```

The **colgroup** element has defined that the first three columns form a group, not a very sensible grouping, and that these should be aligned to the right. The attribute **span** defines the number of columns in the group. Within that group of columns, the two **col** elements define the first column as having a width of 30% and the next two have a width of 10%. For the **col** element, the attribute **span** defines the number of columns that the element applies to. Finally, the second **colgroup** element specifies the group of two columns having a width of 20%.

| W3C Membership | | | | |
|----------------|----------|--------|---------|-------|
| Type | Americas | Europe | Pacific | Total |
| Full | 62 | 29 | 17 | 108 |
| Affiliate | 240 | 123 | 47 | 410 |
| Total | 302 | 152 | 64 | 518 |

The **col** element can be used by itself and appear without being surrounded by a **colgroup** element. The **colgroup** element does not add any additional functionality as it has the same attributes as the **col** element. It does make the description simpler and brings out any inherent structure to the table.

8.4 Cell Organisation

```
<tr>
  <th rowspan= colspan= align=>
  <td rowspan= colspan= align=>
</tr>
```

Frequently in the organisation of more complex tables, it is useful to be able to have a cell in the table that spans more than one column or more than one row. The attribute **rowspan** defines the number of rows that the cell spans (default is 1) and **colspan** defines the number of columns that the cell spans. For example, to have a row in the head of the table to replace the caption, this could be done as follows:

```

<table title="Membership of W3C July 2001" summary="5 ..." width="90%">
<thead>
<tr>
<th colspan="5">W3C Membership</th>
</tr>
<tr>
<th>Type</th><th>Americas</th><th>Europe</th><th>Pacific</th><th>Total</th>
</tr>
</thead>

```

The original table would then appear as:

| W3C Membership | | | | |
|----------------|----------|--------|---------|------------|
| Type | Americas | Europe | Pacific | Total |
| Full | 62 | 29 | 17 | 108 |
| Affiliate | 240 | 123 | 47 | 410 |
| Total | 302 | 152 | 64 | 518 |

To make the first and last titles straddle the first two rows, the table could be changed to:

```

<table title="Membership of W3C July 2001" summary="5 ..." width="90%">
<thead>
<tr>
<th rowspan="2">Type</th>
<th colspan="3">W3C Membership</th>
<th rowspan="2">Total</th>
</tr>
<tr>
<th>Americas</th><th>Europe</th><th>Pacific</th>
</tr>
</thead>

```

Clearly some care must be taken in designing this. The first row consists of three cells. The first cell straddles this row and the next one as does the last. The middle cell straddles three columns. When the next row is to be defined, the first and last cells have already been defined by the previous row. In the second row, only the remaining three cells need to be defined. the result is as follows:

| Type | W3C Membership | | | Total |
|--------------|----------------|--------|---------|------------|
| | Americas | Europe | Pacific | |
| Full | 62 | 29 | 17 | 108 |
| Affiliate | 240 | 123 | 47 | 410 |
| Total | 302 | 152 | 64 | 518 |

9. Forms

- [9.1 Introduction](#)
- [9.2 Form Processing](#)
- [9.3 The input Element](#)
- [9.4 Selections](#)
- [9.5 Writing Essays](#)
- [9.6 Improving Layout](#)
- [9.7 The button Element](#)
- [9.8 Summary](#)

9.1 Introduction

Forms in HTML are much like forms in real life. The user must fill in a set of **fields** on the form and **post** it somewhere. For the moment, we will concentrate on filling the form in rather than posting it. For commercial activities, it is usual for the form to be sent back to the server for processing. However, if the form is just designed to provide interaction between the user and the Web page, it may be sent to a script for execution locally.

Forms in real life come up in many shapes and sizes. Many forms provide text fields for you to fill in (your name and address for example). Some present you with a set of options and ask you to tick **all** that apply and some insist that you only tick one of the options. Most of the standard methods of form filling are provided in HTML. The **form** element encloses the complete form. The **input** element handles most of the fields on the form and the attributes of the element define what type of field it is. A separate **textarea** element is provided for those fields on forms where you are allowed to write essays rather than simple one line answers. The **select** element handles those scrolling lists where you have to select your country of origin etc. Within the list, the individual entries are handled by the **option** element.

So far the form is a set of input fields that are independent. By using other HTML elements such as headings and tables, the form can be made to look as though it has some structure. To aid accessibility, the form has its own internal structure. A set of form fields can be grouped together by the **fieldset** element and this section can be given its own title using the **legend** element. Also, individual fields can have a label associated with them using the **label** element.

To make even prettier buttons, HTML has added the **button** element which forms much the same as the **input** element when it is defining a button. However the **button** element can embed any block-level elements within it including images so it greatly enhances the look of the button but does not really add much content capability.

9.2 Form Processing

```
<form method= action= onsubmit= >  
<input name= type= value= size= maxlength= checked= src= accesskey= >
```

Here is a very simple example of a form:

```
<form method= "post" action = "mailto:bhopgood@brookes.ac.uk">  
<p>Please type the mail to send to Bob Hopgood:  
<input name="MyMessage" type="text" ></p>  
</form>
```

This would appear something like:

Please type the mail to send to Bob Hopgood:

The **input** element will be discussed in detail in the next section. The example here just expects the user to type some information into a box. The **form** element has two major attributes. The **method** attribute indicates the HTTP method to be used in sending the information and the **action** attribute gives details of where it will be sent. In the example, the HTTP POST method is specified and the input is to be sent to the mail address specified. The possible methods are:

get

This is the default action. If a URL is specified as the value of the **action**, a **?** followed by the input is sent to the server or whatever processing agent is specified.

post

In this case the input is embedded in the form and sent to the processing agent

In the example, if the text input is **Hello world** followed by a Return, this will cause the form to be submitted. The content of the form that is emailed is:

```
MyMessage=Hello+world
```

If the **form** element had been:

```
<form method= "get" action = "http://www.w3.org/">
```

The URL requested would have been:

```
http://www.w3.org/?MyMessage=Hello+world
```

If the form has several input values, the individual inputs will be separated with **&**. For example:

```
http://www.w3.org/?MyMessage=Hello+world&MySecondMessage=From+me
```

It is possible to send the input results to a script associated with the web page by:

```
<form onsubmit="process()">
```

In this case the function **process** can access the result of the input and perform the script based on this input.

To a large extent, what happens as a result of the input is outside the control of HTML and we will not discuss it further in this Primer. The aim here is to show the types of input possible and how they can be used.

9.3 The input Element

```
<form method= action= onsubmit= >  
<input name= type= value= size= maxlength= checked= src= accesskey= >
```

The overall form of the **input** element depends on the three attributes **name**, **type** and **value**. The other attributes are either required for a subset of the devices or the meaning varies dependent on the device:

name

The **name** attribute gives the name to be added to the input to indicate where it has come from. In most cases, the input device has an initial value so that if you submit the input without doing anything, this value will be sent to the destination. The input device has the ability to change the initial value and this is usually what will be sent.

type

The **type** attribute defines the type of input. The possible input types are: **text**, **submit**, **radio**, **checkbox**, **button**, **reset**, **password**, **file**, **hidden** and **image**. We shall look at these in detail in this section. If no **type** value is given, the input type is assumed to be **text**. In the example in the previous section, the **type** attribute could have been omitted.

value

The **value** attribute gives the initial value for the input from a device in most cases. The only device that must have an initial value is the **radio** device.

size

This either gives the width of the input device in pixels or the number of characters allowed.

maxlength

For **text** and **password** devices, this defines the maximum size that the input can take in characters.

checked

For **radio** and **checkbox** devices, this says whether the button is set initially.

src

For **image** devices, this points to the image.

accesskey

The **accesskey** attribute can be used more generally than input but it is particularly useful in the environment where an operator has a large number of forms to fill in. It associates a key on the keyboard to the input device and in the case of radio buttons will change the value. For somebody that has accessibility problems it allows them to quickly focus on a specific device where they wish to input a value.

text

The **text** device has already been shown in the previous section but here is a more complete example:

```
<input name = "MyMessage" type="text" value="Some text" size="40"  
maxlength="30" accesskey="Z" />
```

This would appear something like:

The device will appear with a window capable of holding the 40 characters that may be input. If more than 30 characters are input, the rest are thrown away. The initial text appears in the window. If the user does not want to use that, this should be deleted before typing is started. Hitting the character **Z** on the keyboard should move the focus ready to input text in the text field. This is still poorly supported in today's browsers. Also, some browsers manage to miscalculate the width of the text field so that even when the **size** is the same as **maxlength**, the field length is not sufficient to see all the characters. On some browsers, the window can be made a scrolling one by: specifying the **maxlength** greater than the **size**. IE6 does not do this. To be sure of seeing all the characters typed:

```
<input name = "MyMessage" type="text" value="Some text:" size="40"
maxlength="10" /></p>
```

This would appear something like:

Some text:

Attempting to type after the ten characters displayed will not cause any action.

submit

So far, to input the text, it has been necessary to hit the RETURN key which may not be that intuitive. The **submit** type allows a separate device to initiate the submission. For example:

```
<form method= "post" action = "mailto:bhopgood@brookes.ac.uk">
<p>Please type the mail to send to Bob Hopgood:
<input name="MyMessage" type="text" />
<input type = "submit" name="Mail" value = "Send Mail" /></p>
</form>
```

This would appear something like:

Please type the mail to send to Bob Hopgood:

Typing **abc** and hitting the submit button would result in **MyMessage=abc&Mail=Send+Mail** being input. As the input is not just the text input but also the contents of the **submit** button, it is feasible to have more than one **submit** button and process the input text dependent on the button pressed.

radio

Radio buttons can be defined by:

```
<p>Indicate your choice of music:</p>
<p>Jazz: <input type ="radio" name="music" value="jazz"
checked="checked" /></p>
<p>Classical: <input type ="radio" name="music" value="classical"></p>
<p>Folk: <input type ="radio" name="music" value="folk" /></p>
```

This would appear something like:

Indicate your choice of music:

Jazz:

Classical:

Folk:

By giving the three buttons the same name they act as a single device with only one being checked at any one time. The initial setting of the device is with the value **jazz** checked.

checkbox

Checkboxes can be defined by:

```
<p>Indicate all the choices of music you like:</p>
<p>Jazz: <input type = "checkbox" name="music" value="jazz" /></p>
<p>Classical: <input type="checkbox" name="music"
value="classical" /></p>
<p>Folk: <input type="checkbox" name="music" value="folk" /></p>
<input type="submit" name="Choice" value="Choices Made" />
```

This would appear something like:

Indicate all the choices of music you like:

Jazz:

Classical:

Folk:

Choices Made

If jazz and folk have been set followed by hitting the submit button, the input would be: **music=jazz&music=folk&Choice=Choices+Made**

button

Buttons are aimed at providing a method for invoking local scripts. Whereas the **submit** button causes the form as a whole to be submitted, the push button causes a script to be launched:

```
<input type="button" value="Click Me" onclick="processclick()" />
```

This would appear something like:

Click Me

It is usual for the **value** of the click to be the name given to the button.

reset

In the original example of text input, the situation could arise that the input provided by the user is wrong and rather than try and patch it up, the best solution is to start again. This could apply to the other input devices in the form as well. The reset button goes through the form and resets all the input devices to their initial values. For example:

```
<form method= "post" action = "mailto:bhopgood@brookes.ac.uk">
<p>Please type the mail to send to Bob Hopgood:
<input name="MyMessage" type="text" value="Initial text" /></p>
<p><input type = "submit" name="Mail" value = "Send Mail" /></p>
<p><input type = "reset" name="Reset" value = "Reset Text" /></p>
</form>
```

This would appear something like:

Please type the mail to send to Bob Hopgood:

If the initial text is overwritten and the reset button hit, the initial text will reappear.

password

Sometimes the text to be input is information that you do not wish somebody else to see who is looking over your shoulder. In this case it would be preferable for the text not to be visible:

```
<p>Please type the mail to send to Bob Hopgood:
<input name="MyPassword" type="password" value="mypass" /></p>
```

This would appear something like:

Please type your password:

As the user types, a different character is presented (often a *) so that the input cannot be seen. Even the initial value is masked in this way.

file

Frequently, the input required is the name of a local file name. In this case it would be useful if the user could browse his filestore and indicate the file to be sent rather than typing the filename. this is achieved by:

```
<p>Input name of picture to be processed:
<input type="file" name="imagefile"
value="C:\Files\test\photo.jpg" /></p>
```

This would appear something like:

Input name of picture to be processed:

The browser may or may not put the initial file value in the window (IE6 does not and Opera does). What it puts on the button that allows you to search the filestore is also up to the browser.

hidden

Sometimes there is a need to send some information to the server that the form designer does not input from the user to fill in. For example, a form may have been sent already from this user and the current form being input needs to be associated with that form. This can be achieved as follows:

```
<input type="hidden" name="from" value="frah" />
```

Nothing appears on the form but when the form is submitted, one of the input values will be **from=frah**.

image

Buttons can be defined by:

```
<p>Please type the mail to send to Bob Hopgood:  
<input name="MyMessage" type="text" />  
then mail by pressing:  
<input type="image" name="Pretty" src="frah.gif" /></p>
```

This would appear something like:

Please type the mail to send to Bob Hopgood: then mail by pressing: 

What gets sent when the form is submitted depends on where the user has hit on the image. It would be something like: **Pretty.x=73&Pretty.y=122**. It is possible to associate a **usemap** attribute to the input element so that different responses could be made dependent on the input values provided.

9.4 Selections

```
<select name= size= multiple= >  
<option value= label= >  
<optgroup label= >
```

A frequent input requirement is to present a set of **options** to a user and give the user the opportunity to select one. The two elements **select** and **option** provide that facility. For example:

```
<p>Please make your choice of music:</p>  
<select name="music" size="6" multiple="multiple">  
<option value = "jazz">Jazz</option>  
<option value = "classical">Classical</option>  
<option value = "folk">Folk</option>  
<option value = "pop">Pop</option>  
<option value = "world">World</option>  
<option value = "zydeco" selected="selected">Zydeco</option>  
<option value = "country">Country</option>  
<option value = "fado">Fado</option>  
</select>
```

This would appear something like this:

Please make your choice of music:



The **select** element is designed to display a menu of options to the user. The **name** attribute gives the name for the information to be returned and the **size** attribute defines the number of options that would be visible initially. The **multiple** attribute defines whether one or more choices are allowed to be made. If the attribute is set, **holding the control key down** and making a selection allows several selections to be made. The **option** elements appear inside the **select** element and each defines one of the entries on the menu. If the **select** attribute is set, that option will be set when the menu is displayed. In this case **zydeco** is set. If jazz was also selected, the value returned would be: **music=jazz&music=zydeco**.

For very long lists, the amount of scrolling that has to be done is excessive. The normal solution to this is to have a main menu that defines groups of menu items and moving right from a group entry gives you the sub-group. The **optgroup** element has been added to provide this additional single level of structure (the element cannot currently be nested). To define two groups would be provided by:

```
<p>Please make your choice of music:</p>
<select name="music" size="4" multiple="multiple">
  <optgroup label="Old">
    <option value = "jazz">Jazz</option>
    <option value = "classical">Classical</option>
    <option value = "folk">Folk</option>
    <option value = "pop">Pop</option>
  </optgroup>
  <optgroup label="New">
    <option value = "world">World</option>
    <option value = "zydeco" selected="selected">Zydeco</option>
    <option value = "country">Country</option>
    <option value = "fado">Fado</option>
  </optgroup>
</select>
```

The initial menu is a 2-item menu displaying **Old** and **New**. By selecting one or other and moving right brings up the sub-menu with the options in the group. The element does not provide any richer input content but does improve the user interface. It is not implemented in IE6.

If the length of the menu is sufficiently small that the **size** attribute can be the length of the options, the display is likely to be as follows (we have limited the options to 6):

Please make your choice of music:



9.5 Writing Essays

```
<textarea name= rows= cols= readonly= />
```

If a body of text is to be input, the **input** element does not provide the real estate for a large volume of text providing just a single line buffer. To provide an area consisting of several lines requires the **textarea** element. For example:

```
<textarea name="para" rows="7" cols="60" ></textarea>
```

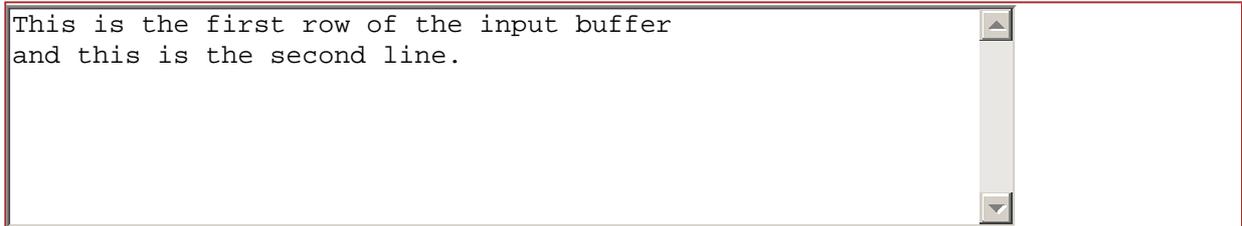
This would appear something like this:



A window containing 7 lines of text, each 60 characters wide is presented to the user and the user can scroll that window so that input is unlimited. To provide an initial value for the input buffer requires:

```
<textarea name="para" rows="7" cols="60" readonly="readonly" >  
This is the first row of the input buffer  
and this is the second line.  
</textarea>
```

This would appear something like this:



By setting the **readonly** attribute, the initial text cannot be overwritten. If it does not appear, it can be overwritten.

9.6 Improving Layout

```
<label for= >  
<fieldset >  
<legend >
```

Quite a few of the input methods provide the mechanism for inputting the value but do not add any information concerning that mechanism. In consequence, normal HTML markup has been added to indicate what the input value represents. The **label** element specifies labels for input devices that do not have a label associated with the device. Earlier, for example, a set of radio buttons was defined as:

```
<p>Indicate your choice of music:</p>  
<p>Jazz: <input type ="radio" name="music" value="jazz"  
checked="checked" /></p>  
<p>Classical: <input type ="radio" name="music" value="classical"></p>  
<p>Folk: <input type ="radio" name="music" value="folk" /></p>
```

Using a label, this could be written as:

```
<p>Indicate your choice of music:</p>  
<label><input type ="radio" name="music" value="jazz"  
checked="checked" />Jazz</label>  
<label><input type ="radio" name="music"  
value="classical" />Classical</label>  
<label><input type ="radio" name="music" value="folk" />Folk</label>
```

Indicate your choice of music:

Jazz Classical Folk

The transmitted form is the same in both cases. It is possible to separate the labels from the input devices while keeping the close association between the label and the device. This is frequently the case when styling is being used to improve the layout. Here is a simple example:

```
<p>Indicate your choice of music:</p>  
<label for="jz">Jazz</label>  
<label for="cl">Classical</label>  
<label for="fl">Folk</label>  
<label for="fl">World</label>  
<input id="jz" type ="radio" name="music" value="jazz"  
checked="checked" />  
<input id="cl" type ="radio" name="music" value="classical" />  
<input id="fl" type ="radio" name="music" value="folk" />
```

Indicate your choice of music:

Jazz Classical Folk World

The user now has the choice of clicking on the radio button or the label itself. This is a slightly contrived example so the benefits are not that obvious in this rather simple example. Note that we have put two labels to point to the third button thus lumping world music in with folk.

For long forms with many inputs, some structure can be added to parts of the form using the **fieldset** and **legend** elements. The first groups a set of inputs and the second gives them a title. For example:

```
<fieldset>
<legend>Personal Details</legend>
<p><label><input name ="NAME" type="text" size="40"
maxlength="40" />Name:</label></p>
<p><label><input name ="EMAIL" type="text" size="40"
maxlength="40" />Email:</label></p>
</fieldset>
<p> </p> <fieldset>
<legend>Musical Interests</legend>
<p>Indicate your choice of music:</p>
<label for="jz">Jazz</label>
<label for="cl">Classical</label>
<label for="fl">Folk</label>
<label for="fl">World</label>
<input id="jz" type ="radio" name="music" value="jazz"
checked="checked" />
<input id="cl" type ="radio" name="music" value="classical" />
<input id="fl" type ="radio" name="music" value="folk" />
<p>Indicate why you like this music</p>
<p><textarea name="para" rows="7" cols="60" >abc</textarea></p>
</fieldset>
```

This would be displayed something like:

Personal Details

Name:

Email:

Musical Interests

Indicate your choice of music:

Jazz Classical Folk World

Indicate why you like this music

abc

A box has been added around the two parts of the form and the legend displayed at the top of each part.

9.7 The button Element

`<button name= value= type= >`

The buttons that can be produced using the `input` element are not that glamorous and, in consequence, a `button` element was defined to replace the `input` element of types `button`, `submit` and `reset`. In no type is defined, a `submit` is produced. The functionality is identical to the `input` element except that it can have any content that you like in defining the information to be displayed on the button. For example:

```
<button name="form" value="submit" type="submit" >
<h2>MUSIC FORM</h2>
<h3>Submit to Server Now</h3>
</button>
```

This might appear as:



As you now have complete control over the content of the button, it can be styled in much the same way as a Web page. Here, some styling has been added to the two HTML elements just to give it a bit more of an effect. The user could add images, background colours, and whatever else is required to produce the desired effect. Most browsers have also added some goodies to the button such as giving it a shadow to give the impression that the button has been physically pressed.

9.8 Summary

That completes the description of the forms section of the Primer. It does not give the complete picture. Some of the facilities associated with scripting have been dealt with in a rather simplistic way. This is more appropriate for a detailed discussion concerning scripting. Similarly, there are a number of facilities concerning `focus` to enable a user with accessibility problems to easily move through a form filling in the sections. This is particularly appropriate for users with mobile phones or using a text-to-speech browser. These will be dealt with in the discussion of accessibility.

10. The Document Head

- [10.1 Introduction](#)
- [10.2 The title Element](#)
- [10.3 The meta Element](#)
- [10.4 Scripting](#)
- [10.5 The style Element](#)
- [10.6 The link Element](#)
- [10.7 The base Element](#)

10.1 Introduction

In an HTML document, the **head** element contains a set of elements that are not concerned with the content of the web page but give additional meta information related to styling, what the page contains, how it is related to other pages etc. The set of elements allowed in the head of the document are:

title

The title of the document.

meta

Defines a set of metadata associated with the document. It is of use to browsers, search engines, editors etc.

script

Defines a script to be associated with the Web page. Although not part of the head, the **noscript** element will also be described.

style

Defines a style sheet to be used by the document

link

Defines document relationships

base

resolves relative URLs

10.2 The title Element

```
<title >
```

The **title** element **must** be present in a document for it to be legal HTML. It serves to identify the document and it is normal for the browser to add the title to its own decoration at the top of the browser window. When a user stores a page in a bookmark (or favourites) list, the bookmark will be identified with the title of the document. In consequence, the title chosen should place the document in its context.

10.3 The meta Element

```
<meta name= content= http-equiv= scheme= />
```

The **meta** element is a major way that a search engine locates pages. Browsers are not required to support the **meta** element so there are few hard and fast rules about what is allowed and what it does. In many cases it may be browser or search engine specific. The two attributes **name** and **content** define a property and its value. As HTML does not define any values that the attribute **name** can take, it is largely up to the users of the page to decide what to do with the information. Here is an example of a typical web site aiming to get itself recognised by the search engines (<http://www.equineworld.co.uk>):

```
<html>
<head>
<title>Horses and ponies, UK</title>
<meta name="description" content="Horses and Ponies: about horses,
ponies, horse riding and equestrianism in the UK. Horses for sale,
equestrian news, equine care, horse chat and more.">
<meta name="keywords" content="horses, horse, equestrian, equine,
ponies, pony, horses for sale, uk">
<meta name="robots" content="index, follow">
<!--Horses and Ponies: about horses, ponies, horse riding and
equestrianism in the UK. Horses for sale, equestrian news, equine care,
horse chat and more.-->
<meta name="owner" content="Equine World UK Ltd">
<meta name="subject" content="horses and ponies, uk">
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-
1">
<meta http-equiv="PICS-Label" content='(PICS-1.1
"http://www.icra.org/ratingsv02.html" l gen true for
"http://www.equine-world.co.uk" r (ca 1 lz 1 nz 1 oz 1 vz 1)
"http://www.rsac.org/ratingsv01.html" l gen true for
"http://www.equine-world.co.uk" r (n 0 s 0 v 0 l 0))'>
```

There are two main types of search engines. One set crawl the web looking at pages while the other type tries to get some human cataloguing of the information. Many of the ones used today are a mixture of both approaches. The web crawl involves looking at pages and following links to other pages. The better ones may look at all the text on a page. The worst ones start at the top and may only look at the head of the document. The **meta** element is one of the elements that some search engines look for. But do not assume that by using lots of **meta** elements you are sure to be recognised. Search engine **spamming** is a problem. if you add words to get your pages listed in a search, the search engines will detect that so make sure any words you use in the meta information are appropriate. For a commercial web site, it is essential that for a relevant query, the site gets ranked in the top 10. If you land up on the second page, you are close to death! So target keywords important for your site must be in the title of the page and in the meta information. Search engines have realised that meta information can be rigged and so the use of this information by search engines is getting less and less important. For example, neither Google or Alta Vista use **meta** element information to rank web pages any more. On the other hand, Inktomi does. Most read and use the information in the **meta** elements in some way. The two most important properties defined by the **name** attribute are **description** and **keywords**. If the search engine believes the page is honest, it will use the **description** in place of one that it might try to derive from its own activities. In the example above, the **description** aims to give a good overview of the whole site. It is clear that the site is about both horses and ponies, that people in the UK are most likely to be interested, that you can buy a horse there and so on. The **keywords** property tries to guess the kinds of searches that people who need to find this site might type.

The **robots** property is primarily aimed at stopping robots visiting pages of no value If the **content** is set to **noindex** robots will not visit the page. In the example, the author has requested a robot to index the page and follow any links on the page. You may wonder why the author has put a comment containing all the keywords after the meta information itself. As some search engines do not look at meta information **and** believe information near the top of the page is most important, adding the comment effectively acts as a meta element for those search engines.

Some of the other values of the **name** attribute are:

author

The author's name

generator

The tool used to create the page

formatter

Similar to generator

copyright

Copyright statement

owner

Equivalent to a copyright statement

subject

Similar to title

It is difficult to be precise as anybody is allowed to dream up values and their usefulness is dependent on industry using them as de facto standards. A set of standard terms would have made the meta element more useful.

A good site for further information about search engines and what they do is <http://www.searchenginewatch.com/>.

The attribute **http-equiv** effectively defines an HTTP header which controls the action of the browser receiving the page and is used to refine the information provided by the actual headers in the HTTP. Some servers translate the meta information into actual HTTP headers automatically. In the example above, the meta elements define the content type of the page and its PICS label (this will be covered in the discussion of metadata). Other possibilities might be:

```
<meta http-equiv="expires" content=" Wed 28 Feb 2002 23:59:59 GMT">
<meta http-equiv="Pragma" content="no-cache">
<meta http-equiv="Refresh" content="0;URL=http://www.newurl.com">
<meta http-equiv="Set-Cookie"
content="cookievalue=xxx;expires=Wednesday, 28-Feb-02 18:12:20 GMT;
path="/>
```

The first states when the page is no longer valid. The second stops Netscape caching the information. The third states when the page should be refreshed and the fourth defines a cookie. Fuller details of what is possible is defined in HTTP.

The **scheme** attribute has been added as a qualifier on what the meta information means. For example:

```
<meta scheme="ISBN" name="identifier" content="0-201-17805-2">
```

This states that to interpret the meaning of **identifier** and the number, you need to realise that the scheme being used is the ISBN classifications scheme. By knowing this, the meta information can be correctly interpreted as the book **Raggett on HTML 4**.

10.4 Scripting

```
<script type= src= >
</script >
```

To a large extent, scripting is outside the scope of this Primer. Suffice is to say that scripts can be placed in either the head or the body of the HTML document. Several scripts can be provided at different points in the page but effectively all the scripts are put together as one script associated with the page. Putting the script in the head of the page is more elegant but may not be the best place. Remember that search engines do not normally read the whole page. In consequence if you have a long script associated with your page, it is possible that all the search engine reads is a load of non-English rubbish as far as it is concerned. An example use is:

```
<script type="text/vbscript"  
src="http://www.w3.org/script/script.js" />
```

Here the script is defined external to the document. The type indicates the MIME type of the script

```
<script type="text/javascript">  
. . . </script>
```

In this case the script is enclosed between the start and end tags. The **body** element has an attribute that can cause the script to be executed when the page is loaded or unloaded:

```
<body onload="processscript()">  
<body onunload="processscript()">
```

Most elements have the ability to define a script invocation on being clicked or moved over. For example:

```
<h1 onclick="ps()">ABC</h1>  
<p onmousedown="ps()">ABC</p>
```

The complete set of events are:

onload

Occurs when the browser completes the page load. Only available on the **body** element.

onunload

Occurs when the browser unloads the page. Only available on the **body** element.

onclick

Occurs when the mouse button is clicked over an element.

ondblclick

Occurs when the mouse button is double clicked over an element.

onmousedown

Occurs when the mouse button is pressed over an element.

onmouseup

Occurs when the mouse button is released over an element.

onmouseover

Occurs when the mouse is moved over an element.

onmousemove

Occurs when the mouse is moved while it is over an element.

onmouseout

Occurs when the mouse is moved away from an element.

onfocus

Occurs when an element receives focus. To be described under Accessibility

onblur

Occurs when an element loses focus. To be described under Accessibility

onkeypress

Occurs when a key is pressed and released over an element

onkeydown

Occurs when a key is pressed over an element

onkeyup

Occurs when a key is released over an element

onsubmit

Occurs when a form is submitted

onreset

Occurs when a form is reset

onselect

Occurs when a user selects some text in either the **input** or **textarea** elements

onchange

Occurs when a device loses focus and its value has changed since gaining focus

10.5 The style Element

```
<style type= >
```

The style element defines a stylesheet to be used by the document in much the same way as the **script** element defines a script. An example is:

```
<style type="text/css" media="print"> . . . </style>
```

The **type** attribute defines the type of stylesheet following and the **media** attribute indicates to which media it applies.

10.6 The link Element

```
<link href= type= media= hreflang= charset= />
```

The **link** element has been available in HTML almost since the beginning. The intention was that the element would give information concerning how this page relates to other pages. As two pages can each link to the other, it is not always obvious which is the parent and which is the child. The **link** element was designed for that purpose and the hope was that search engines would use the link information to improve search relevance. To a large extent it is not used in that role. However there are some specific uses of the **link** element:

```
<link href="../../../primerstyle.css" rel="stylesheet" type="text/css"
media="print"/> <link rel="alternate" hreflang="pt" type="text/html" />
<link rel="alternate" hreflang="ar" charset="ISO-8859-6"
type="text/html" />
```

The first example shows a link to an external stylesheet to be used when the document is printed. The second gives an alternative to the current page in Portuguese. The third links to an alternative page in Arabic. In this case, the character set needs to be defined also. Search engines may offer the linked pages when the user has requested pages in a specific language.

10.7 The base Element

```
<base href= />
```

The **base** element provides the base for relative URLs. For example:

```
<base href="http://www.w3.org/nuts/index.htm" />
. . . <a href="../../../brazil/origin.htm" />
```

This would take the relative URL and, as the **base** element is declared, the relative URL will be resolved to:

<http://www.w3.org/brazil/origin.htm>

by effectively treating the current page as though it were the one defined by the **base** element.

11. Creating HTML

The main tool that you need to write HTML is a text editor. There are other tools on the market but most come with some kind of health warning. For example, it is possible to write a Web page using Microsoft Word. Suppose the aim is to produce:

```
<html>
<head>
<title>My First Web Page</title>
</head>
<body>
<h1>My Title</h1>
<p>My first paragraph</p>
</body>
</html>
```

Using Word's built in heading and paragraph styles and saving as a Web page produces:

```
<html xmlns:o="urn:schemas-microsoft-com:office:office"
xmlns:w="urn:schemas-microsoft-com:office:word"
xmlns="http://www.w3.org/TR/REC-html40">
<head>
<meta http-equiv=Content-Type content="text/html; charset=windows-
1252">
<meta name=ProgId content=Word.Document>
<meta name=Generator content="Microsoft Word 9">
<meta name=Originator content="Microsoft Word 9">
<link rel=File-List href="./MyParagraph_files/filelist.xml">
<title>My Paragraph</title>
<style>
<!--
/* Style Definitions */
p.MsoNormal, li.MsoNormal, div.MsoNormal
{mso-style-parent:"";
margin:0in;
margin-bottom:.0001pt;
mso-pagination:widow-orphan;
font-size:12.0pt;
font-family:"Times New Roman";
mso-fareast-font-family:"Times New Roman";}
h1
{mso-style-next:Normal;
margin-top:12.0pt;
margin-right:0in;
margin-bottom:3.0pt;
margin-left:0in;
mso-pagination:widow-orphan;
page-break-after:avoid;
mso-outline-level:1;
font-size:16.0pt;
font-family:Arial;
mso-font-kerning:16.0pt;}
```

```
@page Section1
{size:595.3pt 841.9pt;
margin:1.0in 1.25in 1.0in 1.25in;
mso-header-margin:35.4pt;
mso-footer-margin:35.4pt;
mso-paper-source:0;}
div.Section1
{page:Section1;}
-->
</style>
</head>
<body lang=EN-GB style='tab-interval:.5in'>
<div class=Section1>
<h1>My Paragraph</h1>
<p class=MsoNormal>My first paragraph</p>
<p class=MsoNormal><![if !supportEmptyParas]> <![endif]><o:p></o:p></p>
</div>
</body>
</html>
```

Some Microsoft specific comments have already been removed but the increase in size is from about 120 bytes to well over 1000 bytes, a 10-fold increase in size and time to download. This is a bit unfair as Word was not designed to produce Web pages so maybe it would be better to use Front Page. The result then is:

```
<html>
<head>
<meta http-equiv="Content-Language" content="en-gb">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<meta name="GENERATOR" content="Microsoft FrontPage 4.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<title>My Title</title>
</head>
<body>
<h1>My Title</h1>
<p>My first paragraph</p>
</body>
</html>
```

This is significantly better than Word. The number of characters is now around 330 compared with 120 but it has increased the size and download time by a factor of 3 while adding to the content of the document almost nothing. To be fair, the HTML markup apart from the meta information is quite good with no superfluous information.

There are some specific HTML editors that just produce the HTML mark-up you ask for and nothing more. One of those is HTML-Kit (<http://www.chami.com/html-kit/>). It does not protect the user from writing illegal HTML but does allow you to send the file to a Validator such as Tidy which then updates the file and shows the original and altered files on the screen side by side. It has a number of word processing facilities such as spell checking.

12. Frames

For completeness, it is probably necessary to talk about frames in HTML although they are not part of strict XHTML. Frames were introduced by Netscape as a way of solving the download problems caused by large pages with many images. If a page had a set of buttons for navigation and a set of logos to display, quite a bit of the information on the page would not change from page to page. The solution is to break the browser's window into a set of rectangular areas and allow each to be updated independently.

The HTML document now looks something like this:

```
<html>
<title>The Title</title>
<frameset>
<frame />
<frame />
<frame />
</frameset>
<noframes>
<body>
- - -
</body>
</noframes>
</html>
```

The attributes of the **frameset** define how the window of the display is to be broken up:

- **rows**: defines the number of rows
- **cols**: defines the number of columns

Both attributes have a comma separated list of values that define the size of each row or column respectively. These values can be one of the following types:

- **integer**: defines the size as a number of pixels
- **percentage**: defines the size as a percentage of the available area
- **integer ***: defines the size as a fraction of the available space
- *****: same as **1***

For example:

```
<frameset cols="100, 10% , *, 3*">
```

If the window was 800 pixels wide, this would divide the area into four columns. The first would be 100 pixels wide, the second 80 pixels wide, the third would be 155 pixels and the fourth 465 pixels. The remaining space has been divided up in the ratio 1:3 between the last two columns. If both the **cols** and **rows** attributes are defined, the frames go from left to right and top to bottom. Within the **frameset** element are a sequence of **frame** elements that define the contents of each frame.

The **frame** element defines the source for the contents of the frame and defines a name for the frame. For example:

```
<frameset rows="20%,30%,50%">
<frame name="frame1" src="contents1.htm" scrolling="no">
<frame name="frame2" src="contents2.htm" scrolling="no">
<frame name="frame3" src="contents3.htm" scrolling="yes">
</frameset>
```

The three frames appear side by side with the contents defined by the three HTML files. The third one will have scroll bars while the first two will not.

Frames mainly make sense where the contents of one frame cannot update the contents of another. The obvious example is where, say, the first frame contains a list of links that define the contents of another frame. The links always remain and hitting a link updates the other frame. This is achieved by defining an attribute **target** which specifies the name of the frame to be updated. For example the contents of the first frame might be:

```
<html>
<title>Contents1.htm</title>
<body>
<a target="frame3" href="newcontents3a.htm"> Action a </a>
<a target="frame3" href="newcontents3b.htm"> Action b </a>
<a target="frame3" href="newcontents3c.htm"> Action c </a>
<a target="frame3" href="newcontents3d.htm"> Action d </a>
</body>
```

Whichever link is hit in the first frame results in the contents of the third frame being updated.

Sometimes you want to get out of the frame hierarchy. To aid this, there are some predefined values of the **target** attribute:

_blank

The document is opened in a new window.

_self

The document is opened in the same frame as the link that was made.

_parent

The document is opened in the parent of the current frame.

_top

The document is opened in the complete browser window.

The need for the last of these is because framesets can be defined within framesets so that you may have to go up an unknown number of levels to break out of the current set of frames.

It is difficult to define frames that are accessible. Some organisations like RNIB believe that they should never be used although W3C is not quite so dogmatic. If you can avoid using them do so. The least that should be done is to include a version of the document as a **noframes** element which could be used by Braille, line mode or spoken browsers.

UK Government web sites have the following structure:

```
<html>
<head>
<title>XXX Department</title>
</head>
<frameset rows="40,*">
<frame name="ukonline_toolbar" title="Latest news"
src="http://www.toolbar.e-envoy.gov.uk/standard-toolbar.htm" />
<frame name="ukonline_content" title="home page of XXX"
src="XXXHome.htm" />
<noframes>
<body>
<p>Welcome to the XXX web site
<a href="XXXHome.htm">Enter the site</a>
or go to the <a href="http://www.ukonline.gov.uk">UK online Citizen
Portal</a>.</p>
</body>
</noframes>
</frameset>
</html>
```

This adds a UK Government Logo and ticker tape news feed to all Government web sites in the top 40 pixels leaving the remainder for the Department's own content. The noframes version allows you to enter the site directly ignoring the ticker tape feed if you do not require it.

If you do not need the whole frame mechanism and just want to insert a single frame in an existing page, this can be achieved using the **iframe** element. For example:

```
<iframe name="inframe" width="240" height="250" scrolling="no"  
src="page.htm" >
```

Links can be targetted at an iframe in the same way as a normal frame. It does reduce the frame overhead for simple usage.

Appendix A

References

There are some useful Web sites and books relevant to HTML:

1. <http://www.w3.org/MarkUp/>
The W3C HTML Web Site which contains up-to-date links to anything relevant to HTML.
2. <http://www.w3.org/TR/html4/>
HTML Level 4.01 Recommendation, the latest version of HTML, 24 December 1999.
3. Ragget on HTML4 (2nd Edition), Dave Raggett, Jenny Lam, Ian Alexander, Michael Kmiec
Addison Wesley, 1998.
4. Big Book of World Wide Web Recs, Peter Loshin
Morgan Kaufman, 2000.
5. Web Protocols and Practice, Balachander Krishnamurthy, Jennifer Rexford
Addison Wesley, 2001.
6. XML How to Program, Deitel, Deitel, Nieto, Lin and Sadhu
Prentice Hall, 2000.
7. <http://www.w3.org/People/Raggett/tidy/>, W3C's Tidy Tool that produces legal HTML and tidies up the document
8. <http://validator.w3.org/>, W3C's Validation Service for Web Pages
9. <http://www.w3.org/TR/xhtml1/>
XHTML Level 1.0 Recommendation, a reformulation of HTML as an XML Application, 26 January 2000

B: Quick Reference Guide

This Appendix lists all the elements available with the following information:

1. **Element name:** if the name is **bold** it signifies that all the global attributes are allowed.
2. **Type:** type of element. Possibilities are block-level, inline and ...
3. **Attributes:** lists attributes specific to this element and also which other more general attributes are allowed.
4. **Allowed in content:** those elements that can be contained directly by this element

The content allowed within an element has the following abbreviations:

[block-level]

p h1 h2 h3 h4 h5 h6 ul ol dl div pre hr blockquote address fieldset table form

[inline]

a br span bdo object img map em strong dfn code q sub sup samp kbd var cite abbr acronym

[empty]

Element has no content

Two elements are allowed anywhere **script, noscript**

Element Name	Type	Attributes	Allowed in content
a	inline	charset type name href hreflang rel rev accesskey shape coords tabindex onfocus onblur	[inline] but not a element
abbr	inline		[inline]
acronym	inline		[inline]
address	block-level		[inline]
area		shape coords href alt	[empty] alt is required
base	head	href id	[empty] only one allowed
blockquote	block-level	cite	[block-level]
body	block-level	onload onunload	[block-level]
br	inline	id class style title	[empty]
button	inline	name value type disabled tabindex accesskey onfocus onblur	[block-level] [inline]
caption			[block-level] [inline]
cite	inline		[inline]
code	inline		[inline]
col		align span width	[empty]
colgroup		align span width	col
dd	block-level		[block-level] [inline]
dfn	inline		[inline]

Element Name	Type	Attributes	Allowed in content
div	block-level		[block-level] [inline]
dl	block-level		dt dd
dt	block-level		[inline]
em	inline		[inline]
fieldset			[block-level] [inline] form legend
form	block-level	action method enctype onsubmit onreset accept accept-charset	[block-level] form not allowed inside
h1 to h6	block-level		[inline]
head	head	lang xml:lang dir xmlns	script style meta link object title base (Only one title and base)
hr	block-level		[empty]
html	root	lang xml:lang dir xmlns	head body
img	inline	src alt longdesc height width usemap	[empty]
input	inline	name type value size maxlength checked disabled readonly src alt usemap tabindex acceskey onfocus onblur onselect onchange accept accesskey	[empty]
label		for accesskey onfocus onblur	[inline] label
legend		accesskey	[inline]
li	block-level		[block-level] [inline]
link	head	charset href hreflang type rel rev media lang xml:lang dir xmlns	[empty]
map		name	[block-level] id attribute is required
meta	head	http-equiv name content scheme id lang xml:lang dir xmlns	[empty]
noscript	block-level inline		[block-level]
object	inline	data type standby height width usemap	[block-level] [inline] param
ol	block-level		li
optgroup		disabled label	option
option		selected disabled label value	text only

Element Name	Type	Attributes	Allowed in content
p	block-level		[inline]
pre	block-level	xml:space	[inline] (Not allowed: img object sub sup)
q	inline		[inline]
samp	inline		[inline]
script	head block-level inline	charset type src defer xml:space	only text
select		name size multiple disabled tabindex onfocus onblur onchange	optgroup option
span	inline		[inline]
strong	inline		[inline]
style	head	type media title xml:space	text only (See CSS for text syntax)
sub	inline		[inline] (Not allowed in pre)
sup	inline		[inline] (Not allowed in pre)
table	block-level	summary width border frame rules cellpadding cellspacing	caption col colgroup tthead ttbody tr (caption followed by col and colgroup elements in any order, then tthead, ttbody and tr elements if ttbody omitted)
tbody			tr
td		align rowspan colspan valign	[block-level] [inline]
textarea		name rows cols disabled readonly tabindex accesskey onfocus onblur onselect onchange	text only
tfoot	align valign		tr
th		align rowspan colspan valign	[block-level] [inline]
tthead	align valign		tr
title	head	lang xml:lang dir xmlns	text only (only one allowed)
tr	align valign		td th
ul	block-level		li
var	inline		[inline]

C: Deprecated Attributes

This Appendix lists the deprecated attributes and the elements to which they apply. These will still appear in HTML documents but their function can be achieved by styling. Future versions of HTML are likely not to support them.

Element Name	Deprecated Attributes
address	align
blockquote	align
body	background text link vlink alink bgcolor
br	clear
caption	align
col	align
colgroup	align
dd	align
div	align
dl	align
dt	align
form	align
h1 to h6	align
hr	align noshade size width
html	version
img	hspace vspace border align
input	align
legend	align
li	type value compact
noscript	align
object	hspace vspace border align
ol	type start compact
p	align
pre	width
script	align language
table	align bgcolor
tbody	
td	nowrap width height bgcolor
tfoot	
th	nowrap width height bgcolor
thead	
tr	bgcolor
ul	type compact

