csc2406

Web Technology

Faculty of Sciences

Study Book

Written by

Dr. Leigh Brookshaw & Dr. Richard Watson
The University of Southern Queensland

Distributed by

The University of Southern Queensland
Toowoomba, Queensland 4350
Australia
http://www.usq.edu.au

Produced with LaTeX by the author(s) using the Department of Mathematics & Computing (http://www.sci.usq.edu.au) StudyBook class. Adapted from the "refrep" class (part of the "refman v2.0e" package by Kielhorn & Partl) to implement Wendy Priestly's *Instructional typographies using desktop publishing techniques to produce effective learning and training materials,*
http://www.ascilite.org.au/ajet/ajet7/priestly.html.

Table of Contents

Preface

Welcome to the course CSC2406, Web Technology. This course will cover:

- the Extensible HyperText Markup Language (XHTML). This is the embedded language used to format static web pages.
- the HyperText Transfer Protocol (HTTP). This is the language which allows communication between a web browser and a web server.
- MIME typing. This is the convention which allows the client to launch the correct helper application or plug-in to process or display a file.
- server and client side image maps, JPEG, GIF and PNG image formats. Image maps allow hyperlinks to be attached to various regions of an image. When designing a site it is important to be aware of the merits and disadvantages of the two main image formats.
- the XHTML *Forms* tag. Forms pages allow users to submit data to a server.
- PHP scripts to process Forms input. These are the programs run by the server to process the input data from a forms page.
- Server Configuration, that is, the various server options available when constructing a website.
- Server Security, that is, how to keep a server secure but retain the flexibility and access required for the Web. This will include the security required for electronic commerce.

At the end of the course you should have a working knowledge of web client and server interaction, how to configure a web server and how to administer a server with varying security requirements.

To be able to develop the required technical expertise in Web Technology you will be required to install and run a web server. The notes for this course are written with a specific server and operating system in mind. As all of the concepts developed and outlined in this ourse are applicable to all servers and operating systems the choice of one server and operating system should not be viewed as a restriction. Once you understand how one server works, you pretty much understand them all.

This course is an introduction to the key concepts needed to be understood if you wish to run and maintain your own web site.

Department of Mathematics and Computing, USQ
(This file created: June 12, 2012)

Chapter 1 Introduction

This module introduces the basic communication paradigm of the Internet. The basic concepts of protocols, port numbers, IP addressing etc. is required to be able to understand the communication between a web client and a web server.

Chapter contents

1.1 The Internet

The Internet began in the late 1970's as a research project of the United States Department of Defense (D.O.D). The US military were experimenting with wide-area networks to see if it was possible to link computers across the US so that they could continue communicating with each other in the advent of a nuclear war. This was desirable, as most of the early warning systems and defense systems were becoming computerized in the 1970s.

In the mid 1980's the Internet began a period of explosive growth as government agencies, academic institutions, private research laboratories and corporations began to inter-connect their computers in a network that has come to span the globe.

Although the *World Wide Web* in the minds of many people, has become synonymous with the *Internet*, the two are quite distinct. The Internet is a hierarchical conglomeration of connected networks. A network can be as small as two machines connected together or as large as is convenient for efficient communication. At the junction of networks there are dedicated computers called *routers*. It is the routers job (among other things) to decide how to get a piece of data from its source to its destination.

The novelty of the Internet is in its heterogeneity. The machines connected to it range from personal computers to high speed supercomputers to devices that are not normally viewed as computers at all, for example, printers, mobile phones, tablets &c. The way devices are inter-connected are also heterogeneous; they include wireless, optic fibers, micro wave links, communication satellites, coaxial cables, copper wires, telephone lines &c.

1.2 Protocols

There is no point in connecting heterogenous computers if they cannot exchange information. The difficulty of interconnecting computers is they do not all speak the same language.

The one thing that all parts of the Internet have in common is the protocol they use to send information from one machine to another. The protocol used is TCP/IP (Transfer Control Protocol/Internet Protocol). This *language (and grammar)* specifies how two computers can find each other, how they introduce themselves, and how they conduct a conversation. Using TCP/IP any computer can contact any other computer on the Internet and exchange data with it *provided* that

1. it knows the remote computer's address, and
2. the remote computer is *willing* to talk.

The TCP/IP is a low level protocol, it is used to establish the link, set up the line of communication and ensure the data arrives at the destination. TCP/IP has no knowledge of the contents of the data or of high level structures. To TCP/IP *all* data is a linear stream of 8-bit bytes. To use an analogy, the telephone company establishes the line of communication when you dial a telephone number and ensures it remains open and all the data (your voice) arrives at the destination. It is up to you, not the telephone company, to ensure that the data you send is understandable at the other end. That is, you both speak the same language! The TCP/IP will maintain the link and ensure the data arrives intact and in the correct order, something else must ensure that the data has meaning.

Obviously it is useless to develop the infrastructure (the Internet, TCP/IP) to link disparate computers if they find each others languages incomprehensible. With the development of the physical infrastructure for the exchange of data there was a corresponding development in high level data exchange protocols.

One of the oldest, and still one of the most important protocols, is the Simple Mail Transfer Protocol (SMTP). This is the language used by different computers to transfer electronic mail around the world. The protocol allows computers to recognise mail messages, and pass the message onto the recipient.

The program Telnet (and the protocol it uses) was developed to allow users to connect to remote computers and log onto them (if they have a valid account). From the remote machine they can interact with the computer as if they were directly connected to it.

The File Transfer Protocol (FTP) was developed to streamline the retrieval of large files from file archives. If you know the name of the archive machine FTP can be used to search the archive and retrieve the required files. Irrespective of the computer you use the commands used in the FTP protocol are the same for getting files from and putting files onto a remote computer's file system.

The Network Time Protocol (NTP) was developed so that computers could set their internal clocks by connecting to a computer that sets its time from an atomic clock—in other words connect to a computer that possibly has a more accurate idea of the correct time. Secure communication is normally predicated on all machines having the correct time—accurate to milliseconds.

There are many more protocols designed for a variety of useful tasks, (e.g. Archie, Gopher, WAIS, finger, ph, &c.) some have been successful some have not. One major difficulty of these protocols is that each required the user to master a different piece of software, no two with the same interface[1]

1.3 IP Addresses

The telephone company assigns a unique telephone number to every telephone. There are no two telephone numbers that are the same in the world (remember a full telephone number incorporates the country code and area code). The TCP/IP is the same, every machine on the Internet is assigned a unique number, the IP address. IP addresses are 32-bit numbers that are usually written out as four 8-bit numbers, separated by dots. There are approximately 4 billion addresses available, which may seem sufficient, unfortunately this is not the case. There are a number of reasons for this,

- there are reserved addresses for special purposes, such as multicasting.
- more importantly addresses are issued in contiguous blocks, not individually.

The IP addressing is organised hierarchically in a series of networks and subnetworks. Blocks of contiguous addresses are issued to organisations and regional networks, who in turn issue sub-blocks of addresses. For example the University of Southern Queensland has been issued the block of addresses

139.86.1.1 to *139.86.255.255*

Within this range blocks are allocated to different Faculties/Departments by the University.

Organisationally, it's simpler to give blocks of addresses to organisations and allow those organisations to divide them up as they see fit. Technically, it's much easier for network routers to determine how to get data from one address to another when the Internet is organised into a hierarchy of networks and subnetworks.

[1] The variety of interfaces required is not surprising. As some of the tasks are so different similar interfaces would have been difficult. Also the interfaces had to be designed to work on a variety of platforms most command line driven. Its only since the early 90's that Graphical User Interfaces (GUIs) have become ubiquitous.

The IP addressing system described here is known as IPv4, unfortunately all IPv4 blocks have been issued (the last high level block was issued in mid 2011) there are no-longer any blocks available—this does not mean that all IPv4 addresses are in use—they are not–it just means that all organisational level blocks of addresses have been assigned. The exhaustion of IPv4 addresses was recognised 15 years before the last block was assigned. The addressing system IPv6 was proposed which uses 128-bit numbers—enough addresses for the foreseeable future. IPv6 adoption has been slow—the main constraint is replacing old network hardware across the world with new hardware that recognises IPv6 addresing. With the exhaustion of IPv4 addresses the adoption of IPv6 addresing will accelerate.

1.4 Domain Names

The IP address is computer friendly but not people friendly. It is difficult to remember and hard to type. For this reason, as well as an IP address computers are also given a people friendly name. The names are assigned using the distributed hierarchical lookup system known as Domain Name System (DNS). In the DNS each machine has a unique name consisting of multiple parts separated by dots (Not unlike IP addresses, except there is no limit to the number of parts).

Table 1.1: The original Organizational Domain Names

Suffix	Meaning
edu	Educational Institutions
com	Commercial Institutions
mil	Military Establishments
net	Network Provider
org	Non-profit organization

The first part of a DNS name is the machine's name, followed by an hierarchical list of names. The first name is usually an identifier for the department to which the machine belongs. The next is usually an identifier for the organization as a whole. The next identifies the type of organization and the last identifies the country[2].

See Table 1.1 for a list of the original preferred organisational types.

The Mathematics and Computing web server is found on machine

`www.sci.usq.edu.au`.

The machine name is `www`, the department/faculty name is `sci`, the organisation name is `usq`, the organisation type is `edu` and the country is `au`.

[2] Only the United States did not require a country code. As the system was developed in the US a country code was never required it had to be added later as the Internet expanded beyond the US borders.

Compare this with the postal service. The United Kingdom is the only place that does not have the country name on its postage stamps. This is because the postal system was developed in the UK and the need for a country identification was required only when the service went beyond country borders.

See Table 1.2 for some examples of country codes. For a complete list of country codes refer to the ISO 3166 country code list in the course resources directory.

Table 1.2: Some two letter country codes

Code	Country
au	Australia
uk	United Kingdom
ch	Switzerland
jp	Japan
de	Germany

This hierarchical list of domain names has broken down as commercial interests have begun to dominate the Internet. Companies prefer short and easily remembered domain names and are not willing to accept the existing convention. Therefore you will find domain names that do not follow any convention—as of mid-2012 high level domain types can be purchased by any organisation with the money.

An important feature of the DNS is that a single machine can have one or more *aliases* assigned to it in addition to its true name. This feature is widely used to give descriptive names to server machines. For example the Department of Mathematics and Computing at USQ maintains both a web server and an FTP server. The address for the web server is

`www.sci.usq.edu.au`,

the address for the FTP server is

`ftp.sci.usq.edu.au`.

Both names resolve to the same IP address and neither is the machine's true name.

1.5 Port Numbers

When two processes on different computers wish to communicate with each other it isn't enough that they know each others IP addresses. They need a mechanism to be able to rendezvous. As a single machine runs multiple processes and supplies multiple services, an external process needs a mechanism to be able to specify the process it wishes to communicate with on a remote machine.

The mechanism used is *port numbers*. The IP address identifies the machine, the port number identifies the particular process on the remote machine. Ports are identified by a number from 0 to 65,535. Any process that wishes to use a port tells the machine it is running on, to reserve a particular port for its exclusive use. Any external process requesting communications with a port can only talk to the process that has reserved the port. The external process does not need to know the name of the local process only the port number it expects it to be listening on.

Well known ports are those that by convention, have been reserved for use by particular services. Table 1.3 lists some of the reserved ports. Ports 0

to 1023 have been reserved for internet services, all other ports are freely available for anyone to use[3].

Table 1.3: Some common reserved port numbers and the service that uses the port.

Port	Service
22	SSH
23	Telnet
80	HTTP
110	POP3
123	NTP
20	FTP data
21	FTP control
25	SMTP

1.6 Clients and Servers

To establish a communications link between two processes (either on the same machine or on different machines) one process must initiate a connection and the other must accept it. This is accomplished using a *server/client* scheme.

Server When the server starts up it signals the operating system that it is willing to accept connections on a given port. It then waits for the connections. The server starts running first.

Client When a client needs to send information to the server or retrieve information from the server it opens a connection to the known port, and passes information back and forth. When finished the client closes the connection.

Most servers can handle multiple simultaneous incoming connections. They do this by either replicating themselves in memory when a new connection is requested or by cleverly interleaving their communication activity amongst the clients.

The distinction between client and server rests on who initiates the connection and who accepts it. Although the server is normally the information provider this is not always the case. However, it is generally true that the client interacts with the user, processing keystrokes and displaying results. The user interacts with the server through the client, never directly.

Exercise 1.1: A simple and informative way to learn about any protocol is to connect to a server using the `telnet` program and talk to the server directly.

Experiment connecting to mail servers by connecting to port 25

For example try the following command

```
telnet www.sci.usq.edu.au 25
```

[3] On Unix systems ports 0 to 1023 are reserved for services controlled by the superuser.

This command directs `telnet` to connect to `www.sci.usq.edu.au` using the SMTP port 25.

This command assumes you are using a Unix system, but any system running telnet that allows you to specify the port should work. After you have connected try the command

```
EHLO machine-name
```

where `machine-name` is your machine's name.

What happens?

To exit type `QUIT`

An alternative address to try if the one above fails is your ISP's mail server.

1.7 The World Wide Web

In 1989, Tim Berners-Lee[4] and his associates at CERN, the European particles physics center, proposed the creation of a new information system called "WorldWideWeb". The system was designed to aid the CERN scientists with disseminating and locating information on the Internet. Particle physics projects are such huge collaborative efforts that a system was needed to unify all the fragmented information services and file protocols into a single point of access.

Instead of having to invoke different programs to retrieve information via different protocols, users would be able to use a single program, called a *browser*, with a single user interface, that would *understand* the various protocols. The browser had the task of figuring out how to fetch the information and display it.

A central part of the proposal was to use a hypertext metaphor: information would be displayed as a series of resources. Related resources would be linked together by specially tagged words, phrases and images. By selecting one of these hypertext links the browser would download the resource even though it is on a different machine and accessed through a different protocol.

The turning point for the Web occurred in 1993, when the U.S. National Center for Supercomputing Applications (NCSA) released its web browser *Mosaic*. This browser used icons, pop up menus, rendered bit mapped text, displayed images, used color links to display hypertext links and provided support for sounds, animations, and other types of multimedia.

1.8 Uniform Resource Identifier

The Uniform Resource Identifier or URI is an abstract standard system for identifying resources on the Internet. There are currently two types of URI's: the Uniform Resource Locator (URL) and the Uniform Resource Name (URN). In this course we will be interested in URLs only[5].

[4] Tim Berners-Lee is currently the head of the World Wide Web Consortium. The body designed to oversee and direct the future evolution of the Web.

[5] The URN system implements a name server mechanism similar to DNS name servers. A URN is a permanent name for a resource that never changes. The server would

1.8.1 Uniform Resource Locator

A Uniform Resource Locator is a way to tell a browser how and where to find an item of interest on the Internet. The URL is a straightforward way (see table 1.4) to indicate the retrieval protocol, host, and location of an Internet resource.

Table 1.4: The anatomy of an URL.

`http://www.sci.usq.edu.au:80/courses/CSC2406/index.html`

http	www.sci.usq.edu.au	80	courses/CSC2406	index.html
protocol	host	port	path	resource

The first part of the URL (delimited by the colon) specifies the communication protocol, eg. http, ftp, news, mailto, gopher, telnet. If the protocol is omitted then a web browser assumes http. The second part, beginning with the double slash and ending with the single slash is the name of the host machine on which the resource resides. An optional port number can be specified but it is only required if the remote server has been configured to use a nonstandard port. The rest of the URL is the path to the resource. The path format is different depending on the protocol used.

1.8.2 Legal Characters in URLs

Only **some** characters are permitted within URLs. Alphanumeric (ie upper- and lowercase letters and numerals) and the characters $_@,- are all legal. The characters =;/#?:%&+ and the space character are also legal but *have special meanings* (eg. the : is used to delimiter the port number). ALL other characters, symbols etc. are **illegal**.

To include special characters *without* their special meaning or to include illegal characters in a URL they must be **escaped**, using an escape code. The escape code consists of the % character followed by the two-digit hexadecimal code of the character. For example, a carriage return can be placed into a URL using the escape sequence %0D, a space is escaped to %20, and the percent sign by %25. The character codes used in a URL are the ASCII character codes (see the ASCII character codes in the course resources directory) and the 8-bit superset, ISO Latin-1 (see the Latin-1 page in the course resources directory).

An example of using escape codes in a URL:

`/courses/CSC2406/Welcome%20Page.html`

1.8.3 URL Addressing

There are two types of URLs, **absolute** and **relative**. An absolute URL contains **all** the information necessary to locate the resource. For instance

return the actual location of the resource derived from the URN.

Obviously URN's would only be used for resources that would be permanent, that is the resource would always exist but its location might change.

the following are absolute URLs

```
http://www.sci.usq.edu.au/courses/CSC2406/index.html
www.sci.usq.edu.au/courses/CSC2406/closed/Changes.html
www.usq.edu.au/library/
```

Though the last two did not specify the protocol the web client will assume *http*.

In the above examples the machine address, the path and the resource were all specified. Given this information you know exactly where the resource is located, or more importantly the web browser does. What about the following valid URLs

```
/courses/index.html
../closed/Changes.html
appendix/ascii.html
```

How does the web browser know where to look for the resources since the addresses are incomplete? The browser must assume the URL is *relative* to the current resource to fill in the blanks.

The current resource is the resource that contains the relative URLs. For example, if you download the HTML page

```
http://www.sci.usq.edu.au/courses/CSC2406/sb/index.html
```

and it contains the relative URLs above, they are interpreted as having the following absolute URLs:

```
http://www.sci.usq.edu.au/courses/index.html
http://www.sci.usq.edu.au/courses/CSC2406/closed/Changes.html
http://www.sci.usq.edu.au/courses/CSC2406/sb/appendix/ascii.html
```

The leading slash of the relative URL `/courses/index.html`, implies that this URL is missing only the machine name. The leading double dots of the relative URL `../closed/Changes.html`, has the same meaning as in Unix, go up to the parent directory first then down into the directory `closed`.

Note

Only relative links not beginning with a `/` (slash) can be used in assignments for this course. Your assignments will be placed in a different directory on a different server and machine than the one they were written on — only relative URLs not starting with a slash will not break.

Exercise 1.2: After installing your own web server, experiment with creating documents containing absolute and relative links.

You will need to know how to specify relative links for the Assignments.

1.9 Questions

Short Answer Questions

Q. 1.3: Explain the difference between the *Internet* and the *World Wide Web*.

Q. 1.4: In what way is the Internet heterogeneous?

Q. 1.5: Why is the *heterogeneity* of the Internet novel?

Q. 1.6: What is a *protocol*

Q. 1.7: What is TCP/IP? (and I don't mean the definition of the acronym).

Q. 1.8: What is FTP? Why is it necessary to create such a protocol?

Q. 1.9: Describe the advantages of organizing the Internet into a hierarchy of networks and subnetworks. (Why are telephone numbers organized this way?)

Q. 1.10: Explain the difference between a computer's IP address and its domain name.

Q. 1.11: How is a *relative* URL different from an *absolute* URL?

Q. 1.12: Why is it convenient to allow machines to have more then one domain name?

Q. 1.13: Describe the Server-Client method of creating a communication link between two computers.

Q. 1.14: What distinguishes a Server from a Client?

Q. 1.15: Which process does the user usually interact directly with, the Server or the Client?

Q. 1.16: Explain why Port numbers are required and how they are used.

Q. 1.17: When a process *listens on a port* what does this mean? What does this tell you about the listening process?

Q. 1.18: Describe the *hypertext metaphor*.

Q. 1.19: Using the URL of this document describe its different parts.

Q. 1.20: What are *escaped characters* in a URL.

1.10 Further Reading and References

- The definition of URI's, URL's and URN's can be found at the World Wide Web Consortium page

  http://www.w3.org/Addressing/

  on addressing resources.

- The ASCII character codes can be found in the course resources directory.
- The ISO Latin-1 character codes can be found in the course resources directory.

Department of Mathematics and Computing, USQ
(This file created: June 12, 2012)

I. Client Side

Chapter 2 Extensible HyperText Markup Language

This module presents an overview of the Extensible HyperText Markup Language (XHTML). XHTML is a variant of HTML (HyperText Markup Language) that uses the syntax of XML, the Extensible Markup Language. XHTML has all the same elements as HTML 4.0, but with a slightly stricter syntax.

Examples, of XHTML documents can be found in the examples directory for this module on the course web site.

This module is not a reference manual, the XHTML elements and attributes discussed here are not complete, but the main elements of the language are covered.

Chapter contents

2.1 Introduction

The (X)HTML is a *mark up* language designed to allow the author to format information to be displayed by a web browser. The term *mark up* means that the formatting commands of the language are embedded *explicitly* in the text to be displayed.

In the (X)HTML the mark up elements embedded in the text are called *tags*. The mark up elements inserted in the text are ordinary ASCII characters (keyboard characters). This means that an ordinary text editor can be used to create (X)HTML documents.

To distinguish the (X)HTML elements from the text that is to be displayed, *all* (X)HTML elements are delimited by angle brackets. For instance, the command `<title>` is the starting tag for the document `title` element. The text within angle brackets is not displayed by the browser. The browser tries to interpret, as HTML commands, all text found between angle brackets.

(X)HTML elements can also have *attributes*. Attributes are used to modify the behaviour of the element or supply the element with necessary information. For example, `<table id="data">` is the (X)HTML tag, `table` is the (X)HTML element (begin a table), and `id` is the element attribute (optional in this case). Another (X)HTML tag example is, `<img src="images/usq.png">` where `img` is the element, `src` is the attribute and `images/usq.png` is the *value* of the attribute. This attribute is not optional.

Attributes that modify the behaviour of an element (this means most attributes) should be avoided. The correct way to modify the browser's default behaviour when interpreting (X)HTML markup is to use "Style Commands". Style commands will be discussed in detail in the next module.

2.1.1 HTML, XHTML and XML

HTML has been the language of the Web since its inception—why create a new language—XHTML? One of the the reasons that the Web grew exponentially in the nineties is the simplicity of HTML and the fact that browsers will interpret and display any HTML no-matter how badly written. This means that there are a lot of badly written HTML pages on the Web—in fact most of them. When the Web was young this loose approach to HTML helped the Web grow—but now that the Web is mature and there is an awful lot of data out there a loose interpretation of HTML has problems.

For instance, it is extremely difficult to parse HTML documents because they are not consistent—this means

- That HTML pages with errors can be displayed very differently in different browsers. Each browser interprets errors in its own way. This means—
- It is very difficult to mine data from the billions of HTML pages because the language is not consistent. The HTML authors did not realise the pages had errors because the browser they used displayed what they expected to see.

The Extensible Markup Language (XML) is a *meta-language* for describing markup languages. That is, XML provides a basic structure and a set of rules to which a markup language can adhere.

XHTML tries to redress the problems of HTML by enforcing the basic syntax rules of an XML language. These rules are:

- XHTML documents must be well formed—a browser should not display a document that is not well formed. (Unfortunately in practice browsers still try and guess what you meant with your broken XHTML.)
- Element and attribute names must be lowercase.
- End tags are required for non-empty elements.
- Empty elements must consist of a start-tag/end-tag pair or use the shorthand empty element notation. For example, for a horizontal rule `<hr />` is a shorthand alternative to `<hr></hr>`, and for a line break `<br />` is a shorthand alternative to `<br></br>`
- Attribute values must always be quoted either using single quotes (`'`) or double quotes (`"`). Quotes must appear in pairs—start with a single quote finish with a single quote.
- Attributes cannot be used without a value.
- An XHTML namespace must be declared in the root `html` element. (See below for further explanation.)
- The `head` and `body` elements cannot be omitted. (See below for further explanation.)
- The `title` element must be the first element in the `head` element. (See below for further explanation.)

Activity 2.A

All the examples in this module can be found on the course web site. Study the examples and how they are rendered by your browser. All examples in this course have been tested under Firefox only.

2.2 Structure of an XHTML document

An XHTML document is composed of 3 parts

1. a line containing the XHTML version information,
2. a header section delimited by the `head` element,

3. a body, which contains the document's actual content. The body may be delimited by the `body` element or the `frameset` element.

Example 2.1: Here is an example of a simple XHTML document, with all the *required* XHTML elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
      "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>Minimalist XHTML document</title>
</head>
<body>
  Hello World!
</body>
</html>
```

2.2.1 Preamble

The line

```
<?xml version="1.0" encoding="UTF-8"?>
```

tells any parser[1] processing this document that what follows is a document that conforms to the XML specification for mark-up languages.

The `DOCTYPE` statement tells the parser the vocabulary and grammar this XML document will be using. In this case we are specifying "XHTML 1.0 Strict", as defined by the Web Consortium (W3C). The *Strict* means the document strictly conforms to the XHTML specification. The URL http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd is the Document Type Definition or the document that describes the vocabulary and grammar of strict XHTML. An alternative document type is

```
<!DOCTYPE HTML PUBLIC
      "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

a looser definition of XHTML — used as a transitional definition from HTML to strict XHTML.

2.2.2 HTML Element

Attributes: (required element),
`xmlns` (required attribute),
`lang` (required attribute),
`xml:lang` (required attribute)

The XHTML document proper is defined by the `<html>...</html>` pair—all XHTML markup in the document must be found between these tags. They must be the first and last HTML tags in the document.

[1] Your web browser must first "parse" your XHTML page before displaying it. That is, the syntax must be analysed and understood before it can be displayed.

The attribute `xmlns` defines the XML namespace of all the elements and attributes in the document. In Example 2.1 the namespace is declared as `http://www.w3.org/1999/xhtml`. Namespaces are used in XML documents to ensure that parsers don't get confused in documents that contain multiple XML languages—different languages may use the same element name for completely different actions. In this course we will be using two XML languages—XHTML, and possibly SVG. The namespace is used to distinguish between the two different languages.

Example 2.2: In MS-Windows namespaces are used to distinguish files on different volumes. For example, consider the following file paths

`H:\CSC2406\HTML\index.htm`

`K:\CSC2406\HTML\index.htm`

Though the path to the file is identical they are interpreted as different files because the namespaces are different (they are on different volumes).

The attribute `lang` specifies the language in the document for an XHTML parser and the attribute `xml:lang` specifies the language in the document to a generic XML parser (the attribute `xml:lang` is specified in the `xml` namespace which is different to that declared with the `xmlns` attribute.)

2.2.3 HEAD Element

Attributes: (required element)

The `head` section is the first main section of an XHTML document. It contains information for the server and the client and its contents will not be displayed by the web client in the web page.

TITLE Element

Attributes: (required element)

The `title` element can only be contained within a `head` section and is required to be in all XHTML documents. The `title` element specifies the title of the document. This title is normally displayed by the web browser in the frame title of the browser window. It is the only element in the `head` section to be displayed anywhere.

BASE Element

Attributes: `href` (required attribute)

The `base` element is used to specify the *base URL* of all *relative URLs* in the document. Without a `base` element all incomplete URLs or relative URLs are completed using the document's URL. The `base` element overrides this default behaviour.

Example 2.3: An example of using the `base` element.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
```

```
      "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>Using the BASE element</title>
 <base href="http://www.sci.usq.edu.au/courses/CSC2406/" />
</head>
<body>
  As there is a <code>base</code> element in this document
  all relative URLs in this document
  are completed using the  <code>base</code> element
  <code>href</code> attribute.
  For example the link <a href="index.html">index.html</a>
  will load document
  <p>
  <code>http://www.sci.usq.edu.au/courses/CSC2406/index.html</code>
  </p>
</body>
</html>
```

The *relative* URL `index.html` will be mapped to the absolute URL

`http://www.sci.usq.edu.au/courses/CSC2406/index.html`

irrespective of the location of the document containing the link.

Note This element should rarely be used if ever. It should **never** be used in any submitted pages in this course as it will break relative URLs and effectively make your assignment unviewable.

META Element

Attributes: `name`, `content`, `http-equiv`

The `meta` element can be used to include name/value pairs describing properties of the document, such as the author, expiry date, keywords describing the content, the program that generated the document *etc.*

The `name` attribute specifies the property name, such as `author` (person who wrote the document), `description` (brief summary), `keywords` (keywords to be used by search engines), `generator` (program that generated the document). The exact way the `meta` element and property names are interpreted is system dependent.

Example 2.4: An example of using the `meta` element with the `name` and `content` attributes.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
      "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>Using the "meta" element</title>
 <meta http-equiv="Pragma" content="no-cache" />
```

```
 <meta name="author" content="Leigh Brookshaw" />
 <meta name="keywords"
       content="HTML tutorial USQ example
                university academic studybook" />
 <meta name="description"
       content="Example HTML document
                illustrating the META tag" />
</head>
<body>
  How the <code>meta</code> tag is used by either the
  web server or the web browser is system dependent.
</body>
</html>
```

The `name` attribute can be replaced with the `http-equiv` attribute. In this case, if the document is retrieved via HTTP, the HTTP server is supposed to to use the `http-equiv` and `content` attributes to generate an HTTP header (see the HTTP Module 7 for a discussion of HTTP headers).

A quick example, the following `META` tag

```
<meta http-equiv="Expires"
      content="Tue, 20 Aug 2010 14:25:27 GMT">
```

results in the following HTTP header being added to the document

```
Expires: Tue, 20 Aug 2010 14:25:27 GMT
```

Note: Some browsers support the *non standard* use of the `meta` tag to refresh the current page after a specified number of seconds, with the option of replacing it with a different URL. This technique should **not** be used to forward users to different pages, as this makes the pages inaccessible to some users. Instead, automatic page forwarding should be done using server redirects (see section 7.2.2).

LINK Element

Attributes: `href`, `rev`, `rel`

The `link` element is not widely used, but is intended to provide information on how the current document fits into a larger set of documents by specifying a table of contents location, the previous and next documents in the series, an advisory title, *etc.* The idea is that the `rel` attribute gives the *relationship* of the document specified by the `href` attribute, to the current document. The attribute `rev` gives the reverse relationship. The most common types of relationships are `contents`, `index`, `help`, `glossary`, `next`, `previous`. For example:

```
<link rel='Contents' href="toc.html" />
<link rel='Previous' href="doc31.html" />
<link rel='Next' href="doc33.html" />
<link rel='Stylesheet' href="site.css" />
```

An important `rel` attribute keyword we *will* be using is the `stylesheet` keyword, which can be used to refer to an external style sheet. See

section 3 for further information on style sheets and the use of the `link` element.

Comments

Comments can be added to any XML document by surrounding the comment text with the begin comment, end comment tags. The begin comment tag is `<!--`, the end comment tag is `-->`. Any text between these two tags will be ignored by the browser.

Example 2.5: An example of using XML comments.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
      "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>Using XML comments</title>
</head>
<body>
  Comments can be imbedded in a
  an XHTML document and they will
  not be displayed by the browser.
  <!-- One use of comments is to annotate a page -->
  Another use of comments is to comment out text
  and HTML commands without removing them from the
  document.
  <!--
      <p>
      This can be convenient when in the process
      of modifying a document. A way of testing ideas.
      </p>
  -->
  XHTML Commands are ignored within comments.
  <p>
  The comments on this page can be seen by viewing
  the source code of the page. In Firefox/Iceweasel
  use Ctrl-u to view source.
  </p>
</body>
</html>
```

Activity 2.B

There are many (X)HTML WYSIWIG composers, unfortunately all produce appalling (X)HTML. In this course all exercises and assignments **must** be written using a normal text editor or at most a tag editor. This way you will hopefully become familiar the XHTML language and its limitations.

Tag editors are smart text editors that can supply XHTML tag templates, that way you do not have to remember all the attributes of an elements. The editor will supply you with a list.

Under Linux, `bluefish` is one XHTML tag editor.

Under Windows, `notepad++` is a good syntax aware editor—though not a "tag" editor.

Become familiar with the text editor of your choice and become comfortable writing XHTML by hand.

2.2.4 Common Attributes

The elements that make up the body of the document, beginning with the `body` element, have a number of common attributes. These attributes serve the same purpose in every element they are found.

title

This attribute supplies advisory information about an element and may be rendered by the browser as a pop-up "tool tip".

style

This attribute specifies the "style" commands for the element. That is, style commands that change the default look of the element. See the CSS Module §3.

id

This attribute assigns a name to an element. The assigned name must be unique in the document. The unique identifier is used as a selector for style commands. This selector means that style commands will only be applied to this element and no no other. See the CSS Module §3.

This attribute can also be used on any element as an address for a URL. See §2.6.1 and the `NAME` attribute of the `ANCHOR` element.

class

This attribute assigns a class name or set of class names to an element. Any number of elements may be assigned the same class name or names. Multiple class names must be separated by white space characters. This selector means that style commands will be applied to all elements with this class name. See the CSS Module §3.

All the above attributes are optional.

2.2.5 BODY Element

XHTML documents should have exactly one `body` element defining the main contents of the page. The only exception is a document that uses frames (see section 2.3.14).

The `body` element contains two major classes of XHTML elements. The *block-level* elements format a "block" or paragraph of text (see §2.3), and the *text-level* elements are used for formatting words and characters in the text (see §2.4). Block level elements can be nested and can contain text level elements, which can also be nested. Text level elements cannot contain block level elements. Block level elements include headings, paragraphs, lists, tables, forms and horizontal

rules. Text level elements include hypertext links, embedded images, and image maps.

The only attribute that should be used with the `body` element is the `title` attribute.

2.3 Block-Level Elements

Block-level elements format text "blocks" that appear in the `body` portion of an XHTML document. Block-level elements can contain other block-level elements as well as text-level elements, while text-level elements are used within paragraphs of text and thus can only contain other text-level elements.

The default style of block-level elements is to span the entire width of the rendered page.

2.3.1 H1...H6 Elements

`h1` through `h6` are used for document headings, with `h1` indicating the top level section heading, `h2` the first-level subheading, `h3` the second-level subheading and so on.

Most browsers render headings in a bold face, with `h1` the largest and `h6` the smallest. The smaller headings (`h5` and `h6`) should be used with caution. Depending on the browser being used and the user's selection of font sizes, the minor headings may be rendered smaller than the default paragraph text.

Headings are left aligned by default, but centred or right aligned headings can be created by using style commands, modifying the default style.

Example 2.6: An example of the header elements `h1...h6`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
      "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>Example of header elements</title>
</head>
<body>
Samples of the 6 headings using the default style
<h1 title="Main heading">H1 heading</h1>
<h2 title="Sub heading">H2 heading</h2>
<h3>H3 heading</h3>
<h4>H4 heading</h4>
<h5>H5 heading</h5>
<h6>H6 heading</H6>
</BODY>
</HTML>
```

Figure 2.1 shows the rendered page.

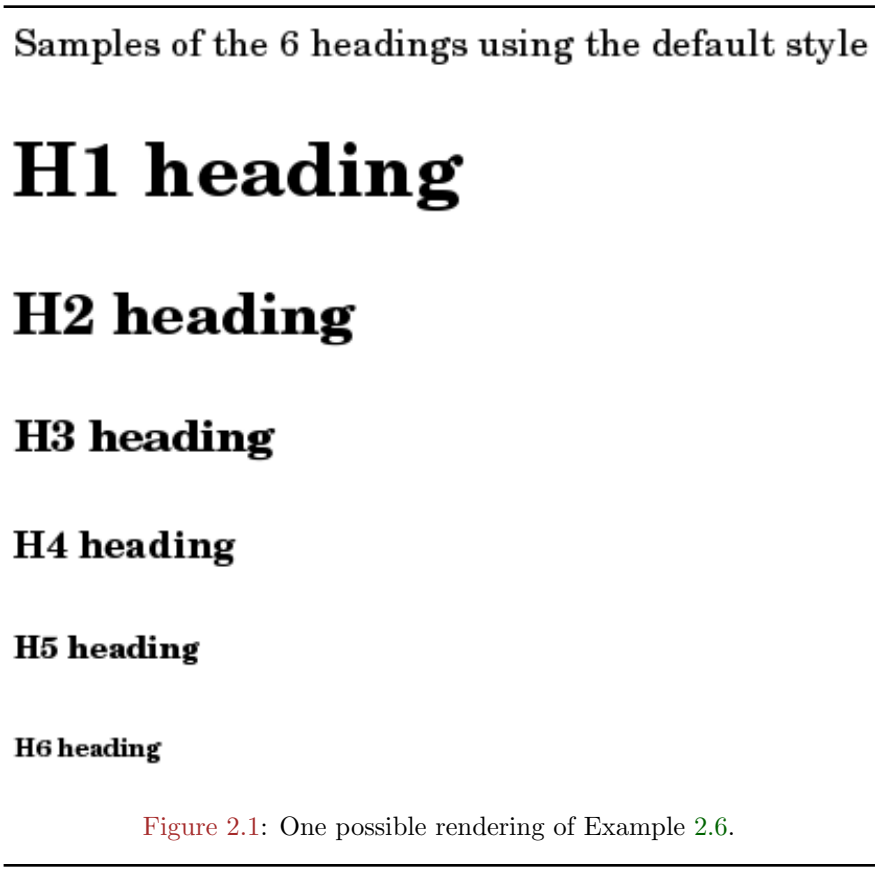


Figure 2.1: One possible rendering of Example 2.6.

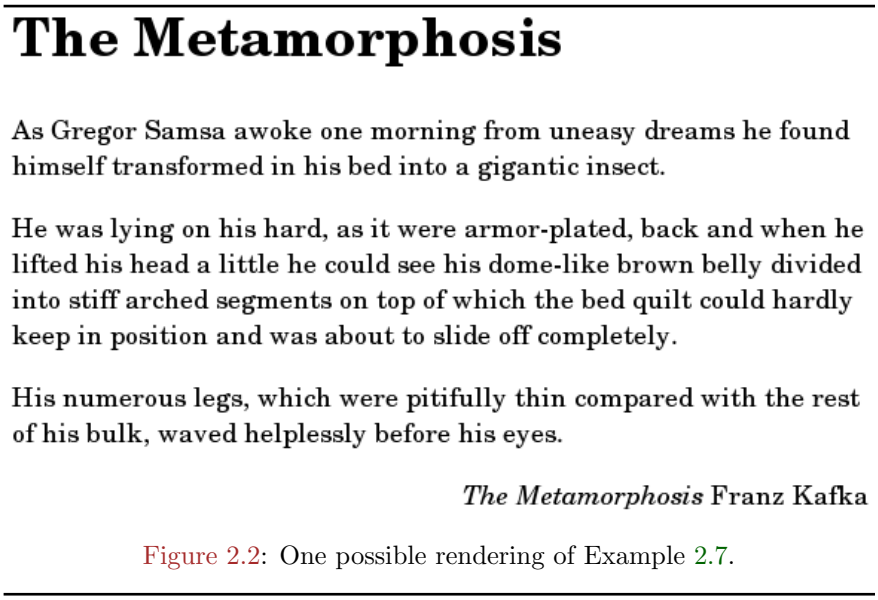


Figure 2.2: One possible rendering of Example 2.7.

2.3.2 P Element

The p element designates basic paragraphs, resulting in a section of text with blank space above and below.

By default paragraphs are aligned left with a ragged right margin. This default behaviour can be modified with style commands.

Example 2.7: An example of the paragraphs.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
      "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>Example of paragraphs</title>
</head>
<body>
<h1>The Metamorphosis</h1>
As Gregor Samsa awoke one morning from uneasy
dreams he found himself transformed in his bed
into a gigantic insect.
<p>
He was lying on his hard, as it were armour-plated,
back and when he lifted his head a little he could
see his dome-like brown belly divided into stiff
arched segments on top of which the bed quilt could
hardly keep in position and was about to slide off
completely.
</p>
<p>
His numerous legs, which were pitifully thin compared
with the rest of his bulk, waved helplessly before
his eyes.
```

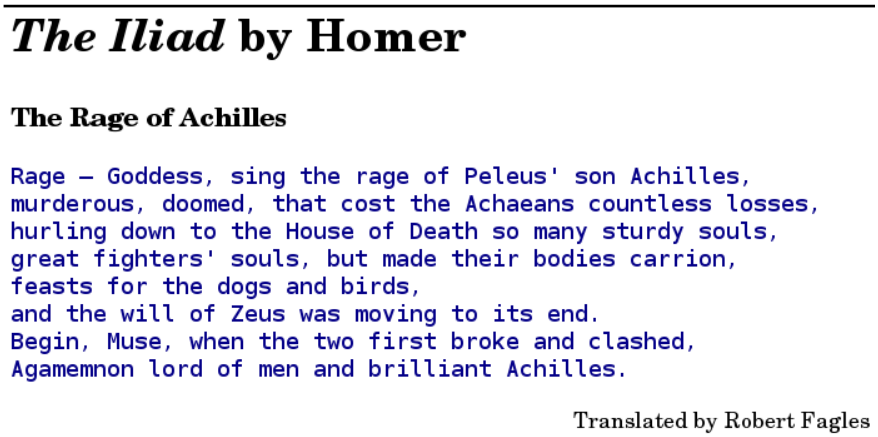


Figure 2.3: One possible rendering of Example 2.8.

```
</p>
<p style="text-align: right;">
<em>The Metamorphosis</em> Franz Kafka
</p>
</body>
</html>
```

Figure 2.2 shows the rendered page.

2.3.3 PRE Element

This element indicates a *preformatted* paragraph that maintains the white space from the source document and uses a fixed width font. Images and elements that *change the font size* are not allowed inside a `pre` container. Although the indentation, blank lines, extra spaces and newlines are maintained, XHTML commands that do not change the font size are interpreted. This means that the `pre` element will not, by itself, be able to display XHTML code uninterpreted.

Example 2.8: An example of the `pre` element.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
      "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>The Rage of Achilles</title>
</head>
<body>
<h1><span style="font-style:italic">The Iliad</span>
     by Homer</h1>
<h3>The Rage of Achilles</h3>
<pre style="color: #040088">
Rage&mdash;Goddess, sing the rage of Peleus' son Achilles,
murderous, doomed, that cost the Achaeans countless losses,
hurling down to the House of Death so many sturdy souls,
```

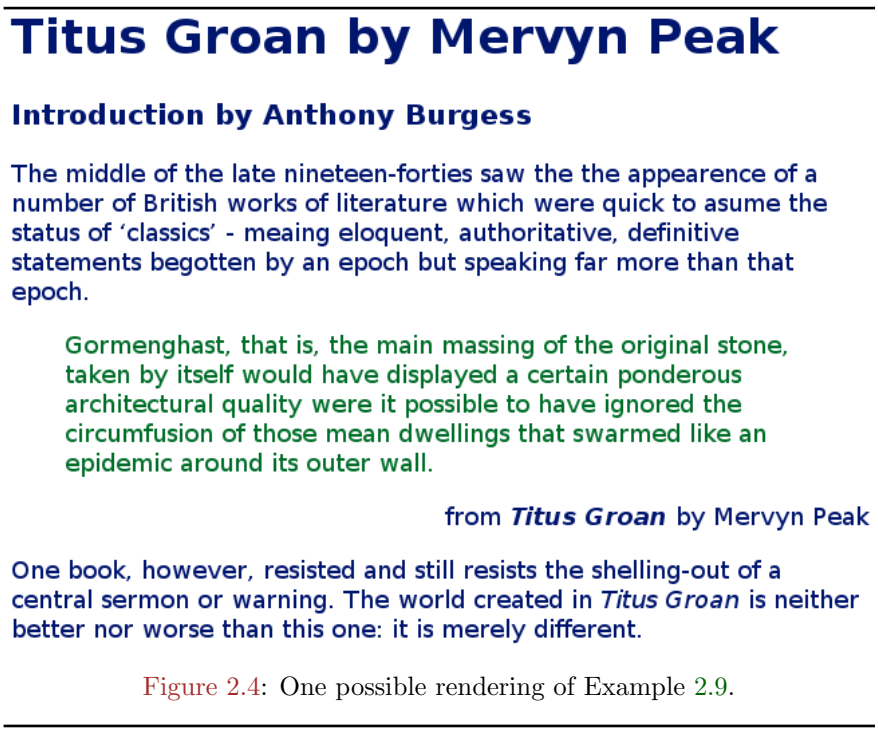


Figure 2.4: One possible rendering of Example 2.9.

```
great fighters' souls, but made their bodies carrion,
feasts for the dogs and birds,
and the will of Zeus was moving to its end.
Begin, Muse, when the two first broke and clashed,
Agamemnon lord of men and brilliant Achilles.
</pre>
<p style="text-align: right;">
Translated by Robert Fagles
</p>
</body>
</html>
```

Figure 2.3 shows the rendered page.

2.3.4 BLOCKQUOTE Element

Attributes: `cite`

This element is intended for large quotations from other works. It is often rendered indented, but this is **not** required. Relying on the web client to indent both left and right margins could be dangerous.

The value of the `cite` attribute is a URL that designates a source document or information about the source.

Example 2.9: An example of the `blockquote` element.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
      "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>Titus Groan</title>
</head>
<body style="color:#00156E">
<h1>Titus Groan by Mervyn Peak</h1>
<h3>Introduction by Anthony Burgess</h3>
The middle of the late nineteen-forties saw the the
appearance of a number of British works of literature
which were quick to assume the status of 'classics'
- meaning eloquent, authoritative, definitive
statements begotten by an epoch but speaking far more
than that epoch...
<blockquote cite="http://www.mervynpeake.org
  style="colour: #006E29">
Gormenghast, that is, the main massing of the original
stone, taken by itself would have displayed a certain
ponderous architectural quality were it possible to have
ignored the circumfusion of those mean dwellings that
swarmed like an epidemic around its outer wall.
</blockquote>
<p style="text-align: right">
from <em>Titus Groan</em> by Mervyn Peak
</p>
<p>
One book, however, resisted and still resists the
shelling-out of a central sermon or warning. The world
created in <em>Titus Groan</em> is
neither better nor worse than this one:
it is merely different.
</p>
</body>
</html>
```

Figure 2.4 shows the rendered page.

2.3.5 ADDRESS Element

The `address` element specifies information such as authorship and contact details for the current document. It should be rendered with paragraph breaks before and after. It usually appears at the top or bottom of the document. Most browsers render the text within the container using an italic font. The browser will wrap the text within the container so line breaks must be explicitly defined (see the Line Break element 2.4.5) if required.

Example 2.10: An example of the `ADDRESS` element.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
      "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>The &ldquo;address&rdquo: element</title>
```

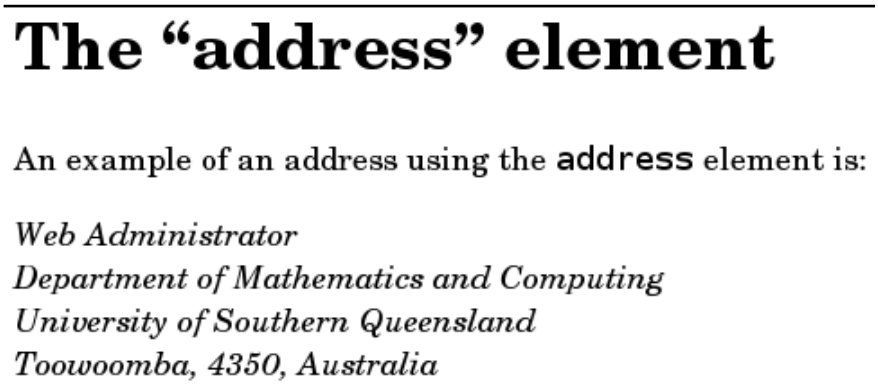


Figure 2.5: One possible rendering of Example 2.10.

```
</head>
<body>
<h1>The &ldquo;address&rdquo element</h1>
An example of an address using the
<code>address</code> element is:
<p>
<address>
Web Administrator<br />
Department of Mathematics and Computing<br />
University of Southern Queensland<br />
Toowoomba, 4350, Australia
</address>
</p>
</body>
</html>
```

Figure 2.5 shows the rendered page.

2.3.6 List Elements

There are a number of commands in XHTML for constructing lists. There are “unordered” lists, “ordered” lists, and “descriptive” lists. Lists can be nested and can appear in table cells.

OL Element

The `ol` element is used to create ordered, or numbered lists. Style commands allow the author to change the numbering style—Arabic, Roman etc., starting numeral, indentation etc.

The default style for numbering is Arabic, which changes depending on the nesting of the ordered list. The default numeral styles are, in order, Arabic, lower Alpha, upper Alpha, lower Roman and upper Roman.

LI Element

The `li` element specifies the individual “list items” in the list. It can contain most other block level elements except for headings and

Example of the OL element

1. BLOCK Level Elements
    1. Headings
    2. Paragraph
    3. Lists
        1. Unordered Lists
        2. Ordered Lists
        3. Definition Lists
    4. Forms
    5. Tables
2. TEXT Level Elements

If you have a look at the source you will see ordered lists inside ordered lists. Note that **all** list items must be terminated correctly.

Figure 2.6: One possible rendering of Example 2.11.

address elements.

Example 2.11: An example of the nesting the `ol` element.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
      "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>Example of "ol" and "li" elements</title>
</head>
<body>
<h1>Example of "ol" lists</h1>
<ol>
  <li> BLOCK Level Elements
     <ol>
         <li> Headings </li>
         <li> Paragraph </li>
         <li> Lists
             <ol>
                <li> Unordered Lists </li>
                <li> Ordered Lists </li>
                <li> Definition Lists </li>
             </ol> </li>
         <li> Forms </li>
         <li> Tables </li>
      </ol>
   </li>
  <li> TEXT Level Elements </li>
</ol>
If you have a look at the source you will see ordered lists
```

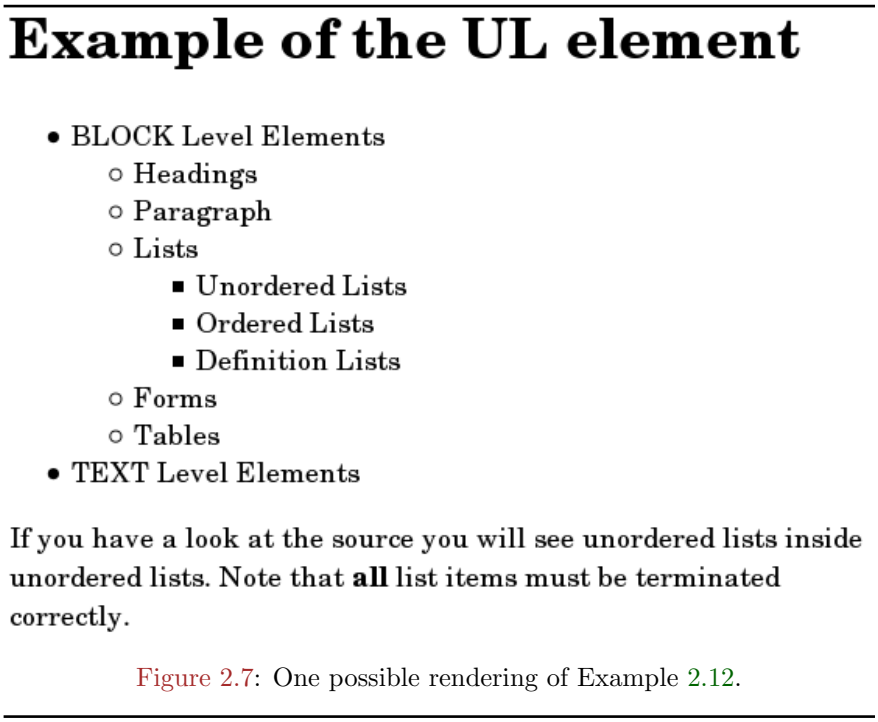


Figure 2.7: One possible rendering of Example 2.12.

```
inside ordered lists. Note that <strong>all</strong> list
items must be terminated correctly.
</body>
</html>
```

Figure 2.6 shows the rendered page.

UL Element

This element is used to create an unordered list or "bulleted" list.

The default style for list items is a "bullet" or solid circle. This changes depending on the nesting of the unordered list. The default bullet styles are, in order, solid disc, circle, square

LI Element

As with the `OL` element, the `LI` element specifies the individual "list items" in the unordered list. It can contain most other block level elements except for headings and address elements.

Example 2.12: An example of the `ul` element.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
      "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>Example of "ul" and "li" elements</title>
```

```
</head>
<body>
<h1>Example of the "ul" element</h1>
<ul>
  <li> BLOCK Level Elements
     <ul>
         <li> Headings </li>
         <li> Paragraph </li>
         <li> Lists
             <ul>
                <li> Unordered Lists </li>
                <li> Ordered Lists </li>
                <li> Definition Lists </li>
             </ul> </li>
         <li> Forms </li>
         <li> Tables </li>
      </ul>
    </li>
  <li> TEXT Level Elements </li>
</ul>
If you have a look at the source you will see unordered lists
inside unordered lists. Note that <strong>all</strong> list
items must be terminated correctly.
</body>
</html>
```

Figure 2.7 shows the rendered page.

DL Element

The `dl` is used for "definitions lists". That is, lists that are in the form of term/definition pairs.

DT Element

This element defines the definition item for the definition list. It can only appear within a `dL` container. The `dt` element should only contain text-level elements.

DD Element

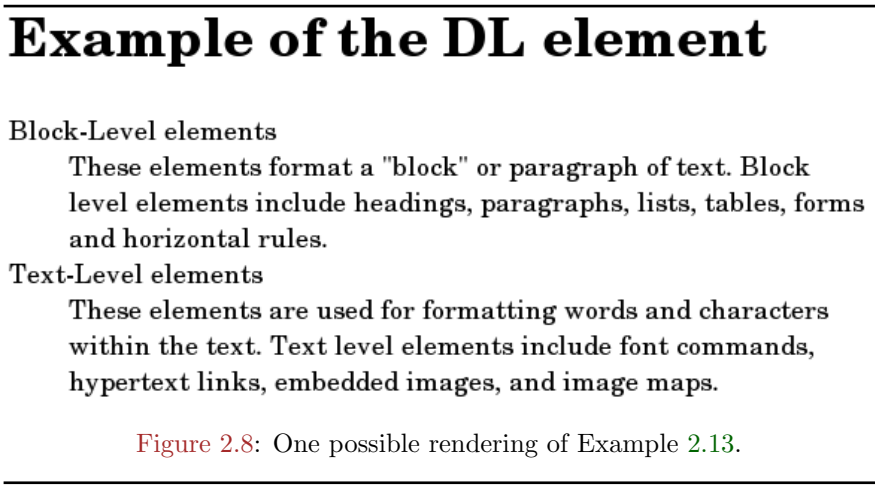
This element defines the definition descriptions for the definition list. It can only appear within a `dl` container.

The `dd` element is allowed to contain other block-level elements except for headings and addresses. A `dd` element can appear in `dl` lists without an associated `dt`.

Note

Because the `DD` elements are usually rendered with an indented left margin, this element is sometimes used to create left indented paragraphs. There is no *requirement* for browsers to indent the left margin, so relying on this behaviour is unsafe. A far better solution is to use *Style Sheets* (see Module 3) in conjunction with the `div` element.

Example 2.13: An example of the `dl` element.



Example of the DL element

Block-Level elements
: These elements format a "block" or paragraph of text. Block level elements include headings, paragraphs, lists, tables, forms and horizontal rules.

Text-Level elements
: These elements are used for formatting words and characters within the text. Text level elements include font commands, hypertext links, embedded images, and image maps.

Figure 2.8: One possible rendering of Example 2.13.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
      "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>Example of "dl", "dt", and "dd" elements</title>
</head>
<body>
<h1>Example of the "dl" element</h1>
<dl>
  <dt>Block-Level elements</dt>
  <dd>These elements format a "block" or paragraph
      of text. Block level elements include headings,
      paragraphs, lists, tables, forms and
      horizontal rules.
  </dd>
  <dt>Text-Level elements</dt>
  <dd>These elements are used for formatting words
      and characters within the text. Text level
      elements include font commands, hypertext links,
      embedded images, and image maps.
  </dd>
</dl>
</body>
</html>
```

Figure 2.8 shows the rendered page.

2.3.7 HR Element (horizontal rule)

Horizontal rules divide sections in a document by drawing a horizontal "grooved" line all the way across the browser window. The style and length of the line can be changed from the default by using style commands.

The Horizontal Rule element is one of two elements (Line Break is the other) where the notion of a container does not make sense. Forcing

all elements to be containers is an XML requirement—so the Horizontal Rule will be en empty container always. It can be written as `<hr></hr>` or more conveniently in the XML shorthand notation for an empty container `<hr />`

2.3.8 Tables

Tables are used to display data in rows and columns of cells. The data to be displayed can be text, preformatted text, images, links, forms, form fields, other tables, etc. In fact, all other elements can be used in tables cells.

The table element is general enough that it was quickly utilised as a layout engine for the entire XHTML page. All of the functionality of the table as a layout engine has been superseded by Cascading Style Sheet's *boxes* and *layers*. Unfortunately using the table syntax is significantly easier than the Cascading Style Sheets syntax as the later is abstract and implementation varies wildly amongst browsers.

Tables are not the best means for laying out an XHTML page because:

- tables are not well defined when rendered by non-visual browsers. For example, an audio browser, braille browser (Cascading Style Sheets have specific commands for the non-visual interpretation of tables).
- tables, especially when cells contain graphics, can force the user to scroll horizontally when viewed on small displays.
- tables with absolute cell widths defined can truncate the cells contents. For example, because the client can choose the font size, preformatted text can become wider than the fixed width cell and will be truncated on the right rendering the contents illegible.

Tables should never be used as a layout engine for entire XHTML pages.

2.3.9 TABLE Element

Attributes: `summary`

The `table` element defines a table and contains all the elements that specify rows, cells, caption etc. of the table.

The attribute `summary` is used to provide a long description of the table that can be used by aural, or braille-based browsers or browsers on hand-held devices with a limited screen size. That is, devices that cannot display the table.

2.3.10 CAPTION Element

This element is used to add a caption to a table. It must appear within the `TABLE` container.

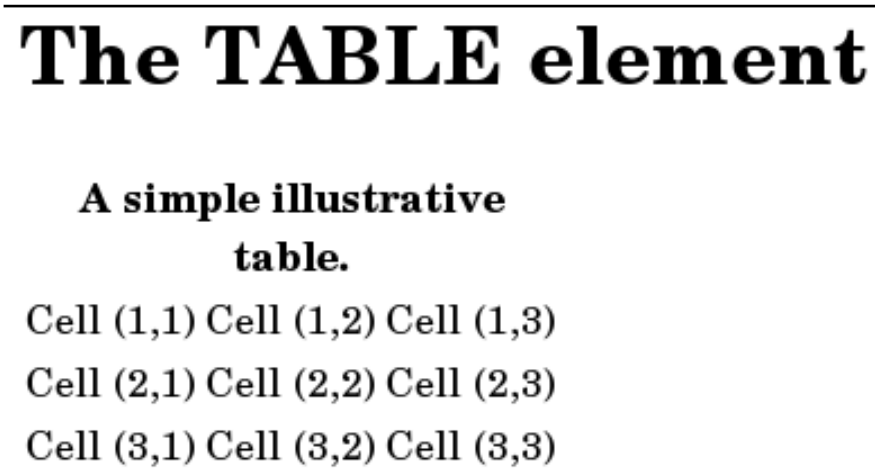


Figure 2.9: One possible rendering of Example 2.14.

The caption by default is placed above the table—this default behaviour can be changed using style commands. Text within the caption are by default centred—this default behaviour can be changed using style commands.

2.3.11 TR Element

This element defines each row of the table. This element can only be used within a `TABLE` container.

2.3.12 TH and TD Elements

Attributes: `rowspan`, `colspan`

These elements specify the data that is to be contained in each cell of the table. These elements can only be used within a `TR` container. The `TH` element is used to specify a *table header* cell. That is, row or column *headings*. The text within the `TH` container is usually rendered in a bold font. The `TD` container specifies a standard *table data* cell.

`ROWSPAN` This attribute specifies how many rows the current cell will span.

`COLSPAN` This attribute specifies how many columns the current cell will span.

Example 2.14: An example of a simple table:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
   "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>A simple table</title>
</head>
```

```
<body>
  <h1>The TABLE, TR, TD elements</h1>
  <table
   summary="A simple illustrative table showing
            the use of the CAPTION,
            TR and TD elements">
    <caption>
        <strong>A simple illustrative table.</strong>
    </caption>
    <tr>
        <td>Cell (1,1)</td>
        <td>Cell (1,2)</td>
        <td>Cell (1,3)</td>
    </tr>
    <tr>
        <td>Cell (2,1)</td>
        <td>Cell (2,2)</td>
        <td>Cell (2,3)</td>
    <tr>
        <td>Cell (3,1)</td>
        <td>Cell (3,2)</td>
        <td>Cell (3,3)</td>
    </tr>
  </table>
<\body>
<\html>
```

Figure 2.9 shows the rendered page.

Example 2.15: An example of a table using the `ROWSPAN` and `COLUMNSPAN` attributes:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
   "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>A cell spanning in a table</title>
 <style type="text/css">
  table {border-collapse: collapse;}
  th,td {border-style: ridge;
         padding: 1ex;}
  caption {caption-side: bottom;}
 </style>
</head>
<body>
  <h1>The ROWSPAN &amp; <br />
      COLSPAN elements</h1>
 <table
    summary="A simple illustrative table showing
             the use of the ROWSPAN,
             and COLSPAN elements">
  <caption>
    <strong>Spanning multiple columns and rows</strong>
  </caption>
  <tr><th rowspan='3'>Span 3<br /> Rows</th>
```

The ROWSPAN & COLSPAN elements

Span 3 Rows	Span 3 Columns		
	Span 2 Columns		Column 3
	Column 1	Column 2	
Row 1	Cell (1,1)	Cell (1,2)	Cell (1,3)
Row 2	Cell (2,1)	Cell (2,2)	Cell (2,3)
Row 3	Cell (3,1)	Cell (3,2)	Cell (3,3)

Spanning multiple columns and rows

Figure 2.10: One possible rendering of Example 2.15.

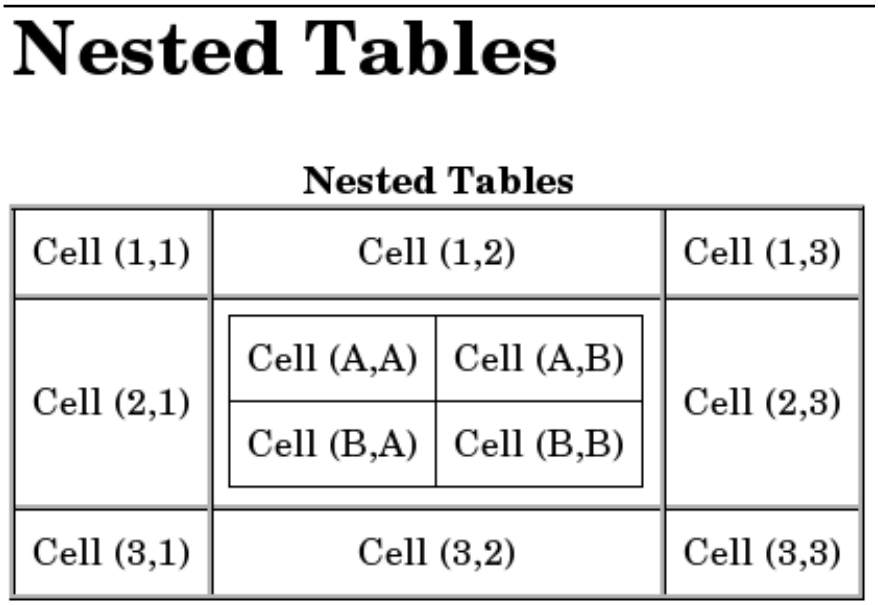


Figure 2.11: One possible rendering of Example 2.16.

```
        <th colspan='3'>Span 3 Columns</th>
    </tr>
    <tr><th colspan='2'>Span 2 Columns</th>
        <th rowspan='2'>Column 3</th>
    </tr>
    <tr><th>Column 1</th>
        <th>Column 2</th>
    </tr>
    <tr><th>Row 1</th>
        <td>Cell (1,1)</td>
        <td>Cell (1,2)</td>
        <td>Cell (1,3)</td>
    </tr>
    <tr><th>Row 2</th>
        <td>Cell (2,1)</td>
        <td>Cell (2,2)</td>
        <td>Cell (2,3)</td>
    </tr>
    <tr><th>Row 3</th>
        <td>Cell (3,1)</td>
        <td>Cell (3,2)</td>
        <td>Cell (3,3)</td>
    </tr>
</table>
</body>
</html>
```

This example uses style commands to add borders to the table and position the caption at the bottom of the table.

Figure 2.10 shows the rendered page.

Example 2.16: An example of a table within another table.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
   "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>Nested tables</title>
 <style type="text/css">
  table {border-collapse: collapse;}
     td {border-style: ridge;
         padding: 1ex;
         text-align: center}

  td table td {border-style: solid;
               border-width: 1px;}
 </style>
</head>
<body>
 <h1>Nested Tables</h1>
 <table
   summary="This is the outside table of an
            example of nested tables">
  <caption>
    <strong>Nested Tables</strong>
  </caption>
  <tr><td>Cell (1,1)</td>
      <td>Cell (1,2)</td>
      <td>Cell (1,3)</td>
  </tr>
  <tr><td>Cell (2,1)</td>
      <td><table
             summary="This is the inside table of an
                      example of nested tables">
            <tr><td>Cell (A,A)</td><td>Cell (A,B)</td></tr>
            <tr><td>Cell (B,A)</td><td>Cell (B,B)</td></tr>
          </table></td>
      <td>Cell (2,3)</td>
  </tr>
  <tr><td>Cell (3,1)</td>
      <td>Cell (3,2)</td>
      <td>Cell (3,3)</td>
  </tr>
</table>
</body>
</html>
```

Figure 2.11 shows the rendered page.

Exercise 2.17: Using a table with "rowspan" and "colspan" attributes produce a timetable page.

Exercise 2.18: Design a simple navigation bar, with about 5 options. Build the bar as 5 separate images and use a table to make the bar appear as one image.

Hint: You will need to set cell padding and borders to zero width—this will require some style commands from the next module.

2.3.13 Forms

These elements let you create data-entry forms, that can be filled out by the user, to be used with server script programming. These XHTML elements are discussed in detail in Module 9

2.3.14 Frames

Frames allow the XHTML author to divide the current window into various rectangular region, each region is associated with a separate XHTML document. Using frames to subdivide a window has some advantages for the author:

- The author can guarantee that certain parts of the page, for example, a table of contents, are always visible on the window.
- The author can avoid reproducing common sections of multiple web documents, by ensuring the same document is included in a frame on each web page.

Unfortunately these are outweighed by the disadvantages for the client:

- The meaning of the "Back" and "Forward" buttons can be confusing to users.
- Poorly designed frames can be disastrous for navigation.
- The address bar of the web client shows the URL of the *top-level document*. This means, it can be hard to find the URL of a particular *frame cell*.
- Some browsers do not support frames (well), and it is difficult to create framed documents that are usable by these browsers.

Note

As most of the author functionality of frames can be replaced with more flexible XML technologies such as XSLT, frames should only be used in specialised situations—not as a general tool for web site development.

In this course Frames should **never** be used!

The Frame Document

In a normal XHTML document, the `BODY` section immediately follows the `HEAD` and contains the body of the web page. In a frames document, the `BODY` is omitted. In lieu of `BODY`, a `FRAMESET` element is used to define the basic row and column structure of the document. For example, the basic frameset document is

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
  <title>Frameset Document</title>
</head>

<frameset ...>
  <!-- FRAME and nested FRAMESET entries -->
  ...
  <noframe>
    <!-- used by non-frame browsers -->
    ...
  </noframe>
</frameset>

</html>
```

Frames are part of the XHTML specification, but the document type description in the `DOCTYPE` declaration needs to change for all `FRAMESET` documents. The *xhtml1-frameset.dtd* is only used for `FRAMESET` documents.

FRAMESET Element

Attributes: `rows`, `cols`

This element divides the current window or frame cell into rows or columns. Entries can be nested, letting the author divide the window into complex rectangular regions.

`ROWS` This element will divide the browser window horizontally. For example:

```
<frameset rows="row1-Size, row2-Size, ..., rowN-Size">
 ...
</frameset>
```

This divides the current window or frame cell into `N` rows. The size of each row is specified in the row list.

`COLS` This element will divide the browser window vertically. For example:

```
<frameset cols="col1-Size, col2-Size, ..., colN-Size">
 ...
</frameset>
```

This divides the current window or frame cell into `N` columns. The size of each column is specified in the column list.

There are three ways to specify the height of a row or the width of a column:

Absolute Size An integer value indicates the size of a cell in "pixels".

Relative Size An integer value followed by a percent sign indicates the size of a cell as a percentage of the "total" window size.

Remainder An asterisk "*"indicates "whatever space remains". the asterisk can be preceded with an integer weight, if there is more than one asterisk entry. For example:

```
<frameset cols="150,20%,*,3*">
 ...
</frameset>
```

indicates there will be four columns. The first will be 150 pixels wide, the second will be 20% of the total width, and the remaining space will be allocated to the final two columns. The first of which will be allocated one quarter of the remaining space, and the other three quarters.

Example 2.19: The following XHTML code is an example of a frameset document.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
  <title>Frameset Document</title>
</head>
<frameset rows="25%,1*,3*">
  <frame src="FrameCell.html" />
  <frame src="FrameCell.html" />
  <frame src="FrameCell.html" />
  <noframe>
    This is an example of frameset document.
    If you are reading this message it means that
    your browser is incapable of displaying frames or
    there is a syntax error in this frameset document.
    Please see the
    <a href="FrameCell.html">non-frames version</a>
  </noframe>
</frameset>
</html>
```

where the `FrameCell.html` document is a standard XHTML page given by—

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
   "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>A Frame Cell Page</title>
 <style type="text/css">
  body {background-color: #700303;
        color: #FFDC4C;}
 </style>
</head>
<body>
```

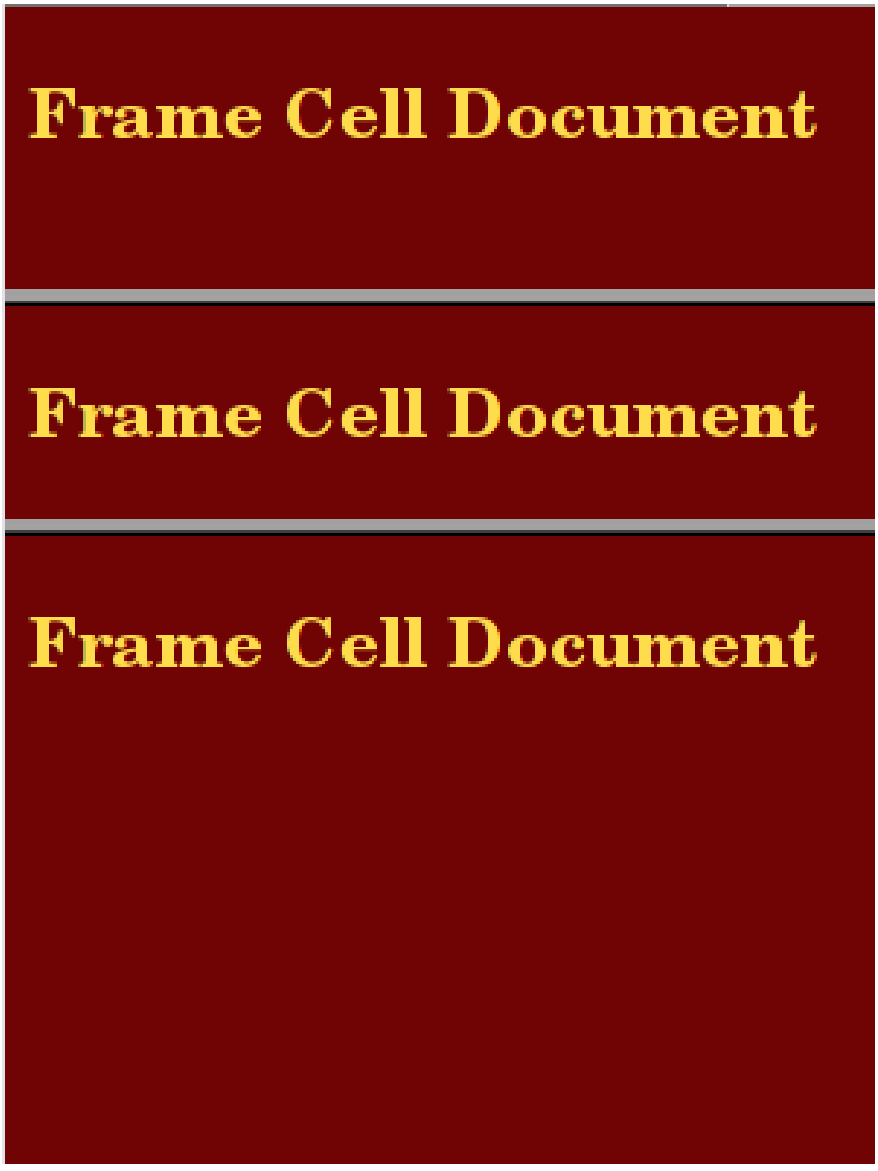


Figure 2.12: One possible rendering of Example 2.19.

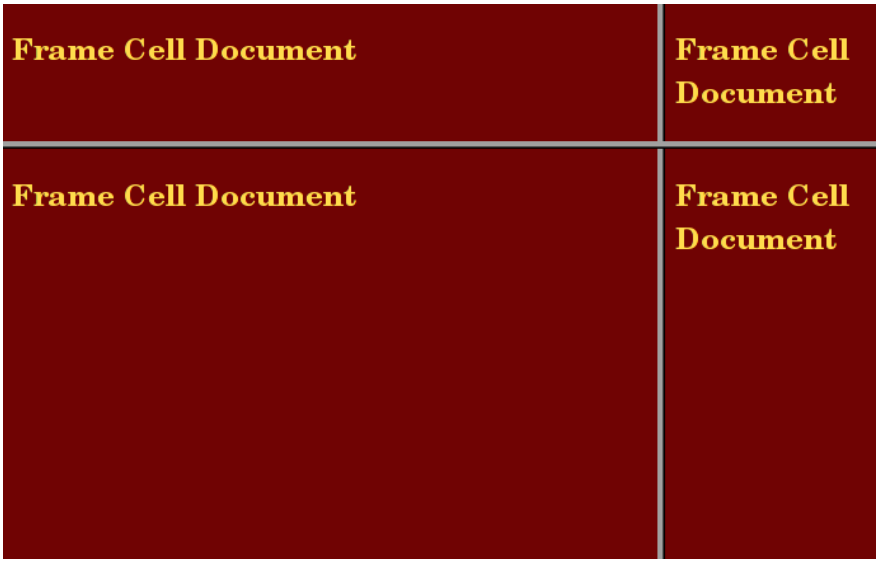


Figure 2.13: One possible rendering of Example 2.20.

```
 <h2>Frame Cell Document</h2>
</body>
</html>
```

Figure 2.12 shows the rendered page.

If both `ROWS` and `COLS` are used within a `FRAMESET` tag then the window is divided into a grid. If the `ROWS` attribute is not set then the columns extend the length of the window. If the `COLS` attribute is not set then the rows extend the width of the window. If neither the `ROWS` or `COLS` attribute are set the frame is exactly the size of the window.

Frames are created left-to-right for columns and top-to-bottom for rows. When both attributes are specified, the page is constructed left-to-right in the top row, left-to-right in the second row, and so on.

Example 2.20: The following HTML code is an example of using both the `ROWS` and `COLUMNS` attributes.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
  <title>Frameset Document</title>
</head>
<frameset rows="25%,*" cols="*,25%">
  <frame src="FrameCell.html" />
  <frame src="FrameCell.html" />
  <frame src="FrameCell.html" />
  <frame src="FrameCell.html" />
```

```
    <noframe>
      This is an example of frameset document.
      If you are reading this message it means that
      your browser is incapable of displaying frames or
      there is a syntax error in this frameset document.
      Please see the
      <a href="FrameCell.html">non-frames version</a>
    </noframe>
  </frameset>
  </html>
```

Figure 2.13 shows the rendered page.

FRAME Element

Attributes: `src`, `name`, `frameborder`, `marginwidth`, `marginheight`, `scrolling`, `noresize`

This element designates the XHTML document that will be placed in a particular frame cell. This element is only legal *inside* the `FRAMESET` container. This element specifies the contents of a frame via its attribute list, therefore it always an empty element so the XML shorthand for an empty element should be employed.

`SRC` This attribute specifies the URL of the document to be placed in the current cell.

`NAME` This attribute gives a name to the current frame. The frame name can then be used as a target for an anchor (see §2.21). The frame name must start with an alphabetic character `[a-zA-Z]`. There are four predefined names that begin with an underscore, see §2.23.

`FRAMEBORDER` This attribute specifies whether the border surrounding a cell will be drawn. The allowed values are 1 (for yes) or 0 (for no). Whether a border appears on the side of a cell depends also on the `FRAMEBORDER` value of the adjacent cells. The `FRAMEBORDER` attribute must be set to 0 on adjacent cells for the border between them not to appear.

`MARGINWIDTH` Specifies the right and left cell margins

`MARGINHEIGHT` Specifies the top and bottom cell margins

`SCROLLING` Specifies whether cells should have scrollbars. There are three possible values. The value `AUTO` means the scrollbar will appear in a cell when needed, this is the default action. The value `YES` means a scrollbar will always appear. The value `NO` means a scrollbar will never appear.

`NORESIZE` By default the user can resize frame cells by dragging the cell border. This attribute disables this default behaviour. As XHTML requires that attributes must have a value the correct usage for `noresize` is

```
<frame src="FrameCell.html" noresize="noresize" />
```

Example 2.21: An example of using nested `FRAMESET`s.

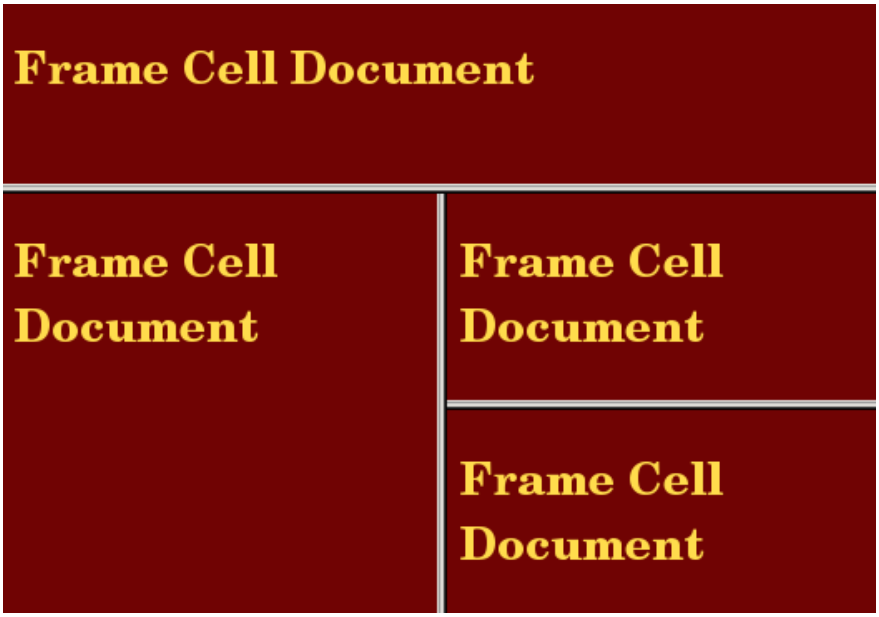


Figure 2.14: One possible rendering of Example 2.21.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
  <title>Frameset Document</title>
</head>
<frameset rows="30%,*">
  <frame src="FrameCell.html">
  <frameset cols="*,*">
    <frame src="FrameCell.html">
    <frameset rows="*,*">
       <frame src="FrameCell.html">
       <frame src="FrameCell.html">
    </frameset>
  </frameset>
  <noframe>
    This is an example of frameset document.
    If you are reading this message it means that
    your browser is incapable of displaying frames or
    there is a syntax error in this frameset document.
    Please see the
    <a href="FrameCell.html">non-frames version</a>
  </noframe>
</frameset>
</html>
```

Figure 2.14 shows the rendered page.

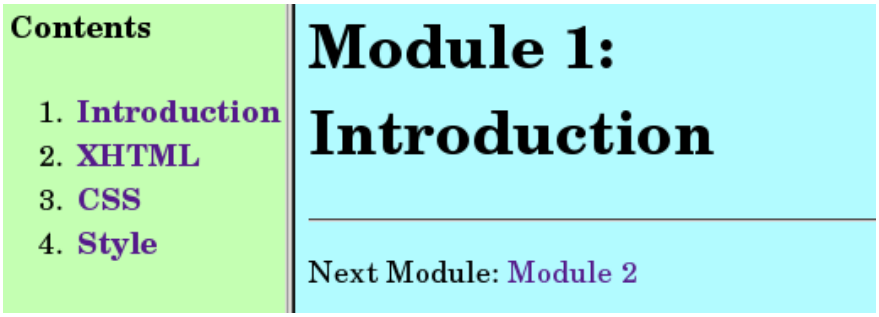


Figure 2.15: One possible rendering of Example 2.22.

NOFRAMES Attribute

All `FRAMESET` documents should contain a `NOFRAMES` container. A browser that supports frames will ignore the XHTML within the `NOFRAMES` container. However, the text will be shown by other browsers, who will ignore all unrecognised XHTML.

The text and mark up inside the `NOFRAMES` container can be used to give a non-frames version of the page, or more probably supply links to a separate non-frame version of the "main" cell of the document.

Authors wedded to frames often don't bother to supply non-frame versions of their documents. This is one of the major reasons frames should be avoided, if you are unwilling to provide a non-frame alternative then don't use frames.

Targeting Frame Cells

A document can specify that pages referenced by hypertext links be placed in certain frames when selected. To do this, the frame cell is given a name via the `NAME` attribute of the `FRAME` element, then the hypertext reference gives a `TARGET` using that name. In the absence of a `TARGET` attribute, the new document will appear in whatever cell the anchor was in. If you specify a target that does not exist, the referenced document is placed in a *new* browser window, which is assigned the target name for future reference. Elements that support the `TARGET` attribute include `A`, `LINK`, `BASE` (defines a default target for the current document), `AREA`, and `FORM`

Note

The `TARGET` attribute can be used to pop up a new browser window (or tag if your browser supports it) without ever using frames and `FRAMESET` documents.

The `TARGET` attribute is one of the only useful ideas to come from frames!

Example 2.22: One common use of frames is to supply a small tool bar or table of contents frame, with a larger region reserved for the main document. Selecting an entry in the table of contents displays the linked document in the main document area.

For example, the following Frameset document:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
  <title>Frameset Document</title>
</head>
<frameset rows="25%,*">
  <frame src="ToC.html" name="ToC">
  <frame src="Module1.html" name="Main">
  <noframe>
    This is an example of frameset document.
    If you are reading this message it means that
    your browser is incapable of displaying frames or
    there is a syntax error in this frameset document.
    Please see the
    <a href="Module1.html">non-frames version</a>
  </noframe>
</frameset>
</html>
```

with the table of contents document `ToC.html`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
  <title>Table of Contents</title>
 <style type="text/css">
  body {background-color: #C4FFB2;}
 </style>
</head>
<body>
<h4>Contents</h4>
<ol>
   <li><a href="Module1.html" target="Main">
      <strong>Introduction<strong></a></li>
  <li><a href="Module2.html" target="Main">
      <strong>XHTML<strong></a></li>
  <li><a href="Module3.html" target="Main">
      <strong>CSS<strong></a></li>
  <li><a href="Module4.html" target="Main">
      <strong>Style<strong></a></li>
</ol>
</body>
</html>
```

and the document `Module1.html`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
```

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
  <title>Module 1: Introduction</title>
</head>
 <style type="text/css">
  body {background-color: #B2FBFF;}
 </style>
<body>
 <h1>Module 1:<br/>Introduction</h1>
 <hr />
 <p>
 Next Module: <a href="Module2.html">Module 2</a>
 </p>
</body>

</html>
```

Figure 2.15 shows the rendered page.

In Example 2.22 the `ToC.html` document had the target explicitly defined for each link. The same effect could have been achieved by using the tag `<BASE TARGET="Main">` in the `HEAD` of the document.

The default behaviour for named frames is that linked documents are displayed in the same cell as the document containing the link (cf. the link in `Module1.html`).

Exercise 2.23: Use the `TARGET` attribute of the anchor element to place documents in new browser windows. Do not use frames or `FRAMESET` documents.

Predefined Targets

There are four built-in frame names that can be used when specifying the `TARGET` attribute: `_blank`, `_top`, `_parent`, `_self`. Since user-defined frame names cannot begin with an underscore, these names are guaranteed to have the same interpretation in all frame documents.

`_blank` Using a link target of "`_blank`" causes the linked document to be loaded into a new unnamed window.

`_self` Using a link target of "`_self`" causes the linked document to be displayed in the current frame cell. In the absence of of a `BASE` target the default behaviour is to place the linked document in the cell containing the link.

`_top` Using a link target of "`_top`" causes the linked document to occupy the whole of the current browser window, thus cancelling all other frames.

`_parent` Using a link target of "`_parent`" causes the linked document to be displayed in the immediate `FRAMESET` parent of the current frame. If there are no nested frames then this should be equivalent to `_self`.

Exercise 2.24: Design a web site for a company that manufactures "Thneeds"

> ...
> A Thneed's a Fine-Something-That-All-People-Need!
> It's a shirt. It's a sock. It's a glove. It's a hat.
> But it has *other* uses. Yes, far beyond that.
> You can use it for carpets. For pillows! For sheets!
> Or curtains! Or covers for bicycle seats.
> ...
>
> From "The Lorax" by Dr. Seuss

Design the site using frames. Have a navigation frame on each page. Companies normally list such sections as "products", "about us", "ordering", "help", "home" etc.

The contents of each section does not have to be very detailed. The idea is to create a "template" for a company site using frames.

Exercise 2.25: Redesign the simple navigation through the examples directories using "table of contents" frames.

2.4 Text-Level Elements

Text level elements allow the author to change the rendering of text fragments. These elements come in two flavours, "Logical text elements" and "Font[2] elements". Font elements explicitly change the style of the rendering font. For example, they can change the font to italic, or bold, change the size of the rendering font, or change to a new font family. Logical text elements on the other hand describe the type of the text being rendered and then allow the browser to make the appropriate rendering choice. For example, text can be required to be *emphasised*, or **strongly emphasised**. In a print based medium emphasised text has traditionally been rendered using an italic font, strongly emphasised text a bold font, but in the web environment, where a browser may not be text based, specifying text should be *emphasised* makes more sense than stating text should be *italic*.

Font elements assume the browser is text based and therefore make explicit font commands. This is not the correct way to format an XHTML document, assumptions about the browser should not be made. It is difficult to see how an italic font can be rendered by an audio browser. For this reason font elements are being removed from the language in favour of logical elements and style sheets (see Module 3) and will not be discussed or used in this course.

2.4.1 Phrase Elements

These elements only have the attributes that all XHTML elements have (TITLE, CLASS, ID and STYLE)[3].

[2] A "font" is a complete set of characters in one size and design.
[3] There are others but they are not covered in this course

EM Element

The enclosed text should be emphasised. The default rendering is to use an italic font.

STRONG Element

The enclosed text should be strongly emphasised. The default rendering is to use a bold font.

DFN Element

This element is used for the defining occurrence of a term. There is no consensus how this should be rendered.

CODE Element

This element is used to display segments of computer code. It is typically rendered in a fixed-width (typewriter) font.

SAMP Element

This element is used to display sample program output. It is typically rendered in a fixed-width (typewriter) font.

KBD Element

This element is used to display examples of keyboard input. It is typically rendered in a fixed-width (typewriter) font.

VAR Element

This element is used to display a variable or an argument to a function or procedure. There is no consensus how this should be rendered.

CITE Element

This element is used to display a citation or reference. The default rendering is to use an italic font.

ABBR Element

This element is used to display abbreviations. The TITLE attribute can be used to give a fuller explanation of the abbreviation.

ACRONYM Element

This element is used to display acronyms. The TITLE attribute can be used to give a fuller explanation of the acronym.

2.4.2 Subscripts and Superscripts

These elements only have the attributes that all XHTML elements have (TITLE, CLASS, ID and STYLE).

SUB Element

This element is used to display subscripts—required for some languages. Normally rendered in a smaller font.

SUP Element

This element is used to display superscripts—required in some languages. Normally rendered in a smaller font.

2.4.3 Document Modification

The elements INS and DEL are used to markup sections of the document that have been inserted or deleted with respect to an earlier version of a document.

INS Element

This element is used to display "inserted" text.

DEL Element

This element is used to display "deleted" text. Normally the text is displayed and "struck-through" by a line.

Both elements take the attributes

`CITE` This attribute is intended to point to information explaining why a document was changed.

`DATETIME` The date and time the change was made.

Example 2.26: An example of the phrase or logical text elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>Phrase Text Elements examples</title>
</head>
<body>
<h1>Phrase Text<br /> Elements</h1>
<em>Emphasised Text</em><br />
<strong>Strongly Emphasised Text</strong><br />
<dfn>Definition Text</dfn><br />
<code>Code Text</code><br />
<samp>Sample Text</samp><br />
<kbd>Keyboard Text</kbd><br />
<var>Variable Text</var><br />
<cite>Citation Text</cite><br />
<abbr title="World Wide Web">WWW</abbr><br />
<acronym title="Women's Army Core">WAC</acronym><br />
SUP example: M<sup>lle</sup> Dupont<br />
SUB example: Carbon Dioxide CO<sub>2</sub><br />
```

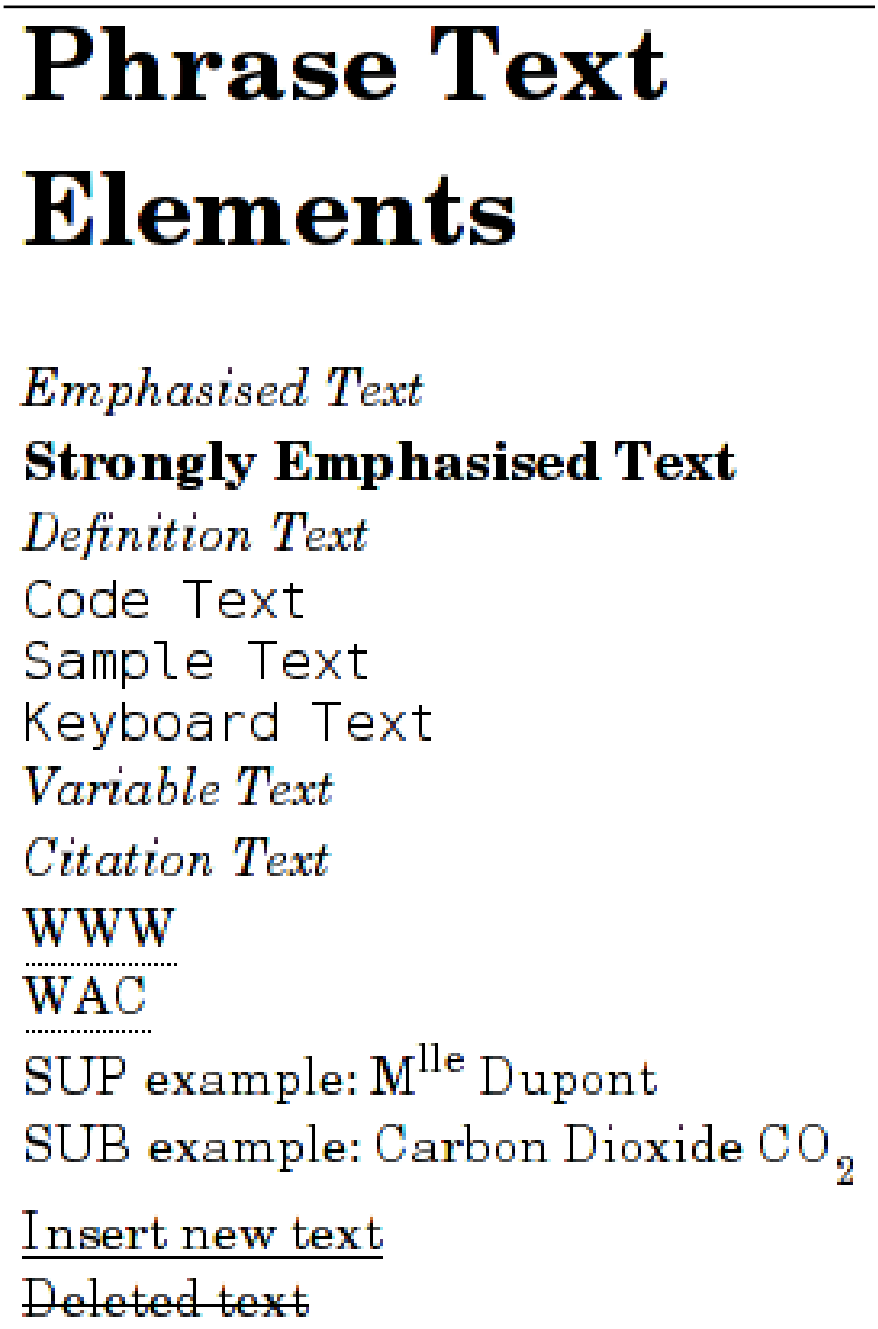


Figure 2.16: One possible rendering of Example 2.26.

```
<ins>Insert new text</ins><br />
<del>Deleted text</del>
</body>
</html>
```

Figure 2.16 shows the rendered page.

2.4.4 Font Elements

As discussed above the font elements are being phased out of the language because these elements only have meaning when the browser is text based. For this reason the use of font elements are discouraged and will not be discussed in this course.

All fonts changes should be done using style sheets.

2.4.5 Controlling Line Breaks

The default action when rendering HTML is to ignore the line breaks within the document (except in the case of text within `PRE` and `TEXTAREA` elements). The browser will insert line breaks into the text based on the width of the rendering window.

BR Element

The `P` element will force a line break but also end the paragraph. The `BR` element inserts a line break without ending the current paragraph.

2.5 Embedded Images

The XHTML allows images to be embedded directly into a document. The format of the image files that can be displayed is dependent on the capabilities of the web client. Most graphical browsers support the Graphics Interchange Format (GIF, pronounced with a hard G), the Joint Photographic Experts Group (JPEG) graphics format and the Portable Network Graphics (PNG, pronounced "ping") format (see Module 4 for more information).

2.5.1 IMG Element

Attributes: `src` (required), `alt` (required), `width` (recommended), `height` (recommended), `usemap`, `ismap`

The `IMG` element is used to insert an image into an XHTML document at the current location.

Note

The `IMG` element is a text level element, and thus does not cause a paragraph break.

`SRC` The required attribute `SRC` specifies the URL of the image file to be inserted into the document. The URL can be either an absolute or relative one. For example:

```
<img src="http://www.sci.usq.edu.au/images/usqweb.gif">

<img src="images/usqweb.gif">
```

ALT This attribute specifies the string to be displayed in place of the image. It is used by text-only browsers, browsers with graphics temporarily disabled, and browsers for people with disabilities that preclude the use of graphics. This is a required attribute.

```
<img src="images/usqweb.gif" alt="USQ Logo">
```

WIDTH

HEIGHT These attributes specify the *intended* size of the image. If the dimensions are different to the original image dimensions, the image will be stretched or shrunk to fit.

Though not required, WIDTH and HEIGHT should be supplied with every image. If the browser knows the width and height of every image on a page, the page can be rendered with text and space left for the images before they have been downloaded. This greatly reduces the time the user must wait before a readable page is available.

```
<img src="images/usqweb.gif" alt="USQ Logo"
     width="116" height="74">
```

Note

Do not use the WIDTH and HEIGHT attributes to scale an image on a page. Forcing a client to download a large image and then only display a scaled version of that image is a misuse of the attributes—and demonstrates a laziness on the part of the page author. Create multiple copies of the image at different resolutions and download those as needed.

USEMAP

IMAP See the graphics module 4, on the use of these attributes.

Example 2.27: An example demonstrating the use of the IMG element can be found in the examples directory for this module on the course web site

2.5.2 Iframe

Attributes: src, name, frameborder, marginwidth, marginheight, scrolling, height,width

The "Inline Frame" element allows authors to insert a frame within a block of text. Using the IFRAME element an XHTML document can be inserted in the middle of another XHTML document. The frame may be aligned with the surrounding text.

SRC This attribute specifies the URL of the document to be placed in the inline frame. (see the attributes of the Frame element.)

NAME This attribute gives a name to the current frame. The frame name can then be used as a target for an anchor (see §2.21). (see the attributes of the Frame element.)

FRAMEBORDER This attribute specifies whether the border surrounding the frame will be drawn. (see the attributes of the Frame element.)

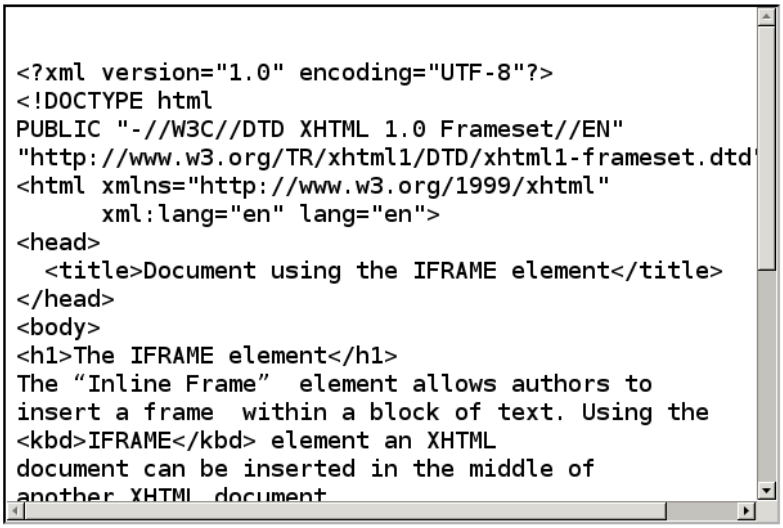


Figure 2.17: One possible rendering of Example 2.28.

`MARGINWIDTH` Specifies the right and left frame margins. (see the attributes of the Frame element.)

`MARGINHEIGHT` Specifies the top and bottom frame margins. (see the attributes of the Frame element.)

`SCROLLING` Specifies whether the frame should have scrollbars. (see the attributes of the Frame element.)

`height` The frame height.

`width` The frame width.

Inline frames may not be resized (so they do not take the `noresize` attribute—as frames do).

Example 2.28: An example of using the `IFRAME` element.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
  <title>Document using the IFRAME element</title>
</head>
```

```
<body>
<h1>The IFRAME element</h1>
The &ldquo;Inline Frame&rdquo;  element allows authors to
insert a frame  within a block of text. Using the
<kbd>IFRAME</kbd> element an XHTML
document can be inserted in the middle of
another XHTML document.
The frame may be aligned with the surrounding
text.
<iframe src="iframe.html" width="400" height="500"
        scrolling="auto" frameborder="1">
[Your web browser does not appear to support frames
 However the source that was to be displayed in this frame
can be <a href="iframe.html">found here</a>]
</iframe>
</body>
</html>
```

Figure 2.17 shows the rendered page.

2.6 Hypertext Links

One of the central ideas of the web and HTML is that documents do not have to be read from top to bottom nor do pages have to be read in sequential order.

Hypertext links can send the client to a new document that has to be downloaded or it can send the client to a new part of the current document.

2.6.1 A Element

Attributes: `HREF`, `NAME` (one or other attribute required)

The text or image enclosed by the `ANCHOR` element becomes a "clickable" region. This region is often underlined and highlighted in the link colour

`NAME` The anchor element can specify a hypertext link or it can specify the end point for a hypertext link.

When the `NAME` attribute is used the anchor element gives the region a *name* so that other links can reference it. It becomes the anchor for a hypertext link. For example:

```
<a name="Beowulf">Beowulf:
         A modern English verse translation</a>
```

It should be noted that the name is case sensitive.

The `NAME` attribute of the anchor element is a way of addressing sections within a document. This means hypertext links can jump into a document, not just to the start of a document.

The same effect can be achieved by any element given a unique name using the `ID` attribute.

```
<h2 id="Beowulf">Beowulf:
        A modern English verse translation</h2>
```

HREF The HREF attribute is used to specify the "address" that the browser should display when the user clicks upon the designated region. The "address" can be an absolute URL, relative URL, an anchor specified with the NAME attribute, or a combination URL and name. For example, using URLs only:

```
<a href="http://www.sci.usq.edu.au/">Department of
                        Mathematics and Computing</a>
<a href="courses/CSC2406">Web Technology</a>
```

To use an anchor specified by the NAME attribute of the ANCHOR element (or the ID attribute of any element) the hypertext reference of the name must be preceded by a # sign. For example:

```
Jump to the <a href="#TOC">Table of Contents</a>
Jump to the <a href="Module1.html#Index">
             Module 1</a> index
```

In the first example, when the "Table of Contents" text is selected then the browser jumps to the region defined by the anchor with the name "TOC" (or for any element with an id attribute with value TOC) within the same document.

In the second example, when the "Module 1" text is selected then the browser loads the resource defined by the URL "Module1.html" and positions the region defined by the anchor with the name "Index" (or for any element with an id attribute with value TOC) at the top of the browser window.

Example 2.29: A large example demonstrating the use of the A element can be found in the examples directory for this module on the course web site

Note When using an image as a link, if the end tag </A> is on a new line and does **not** come immediately after the end of the IMG tag then there could be a small underscore after the image. This is the browser underlining the white space at the end of the link region.

2.7 Unicode

In general any Unicode character can be inserted into an XHTML document—either directly by creating a UTF8 document or by using "character entities". The character entity construct "&#xxxx;" will insert the character with decimal number xxxx at the point it occurs. Values of 0–127 represent the standard ASCII character set, *printable* characters in the range 128–255 represent the standard Latin-1 character set, outside this range the W3C have defined a set of special characters and a set of symbols and Greek characters (All XHTML character codes can be found on the course web site, or follow the links above.).

As well as the number of the character, characters can be referenced using a mnemonic. For example, the copyright symbol ©, can be referenced as "`&#169;`" or using the mnemonic "`&copy;`" in both cases the character entity is bracketed with "`&`" and "`;`".

Table 2.1: Characters codes needed for XHTML special characters

Character	HTML mnemonic	HTML Number
<	<	<
>	>	>
&	&	&
"	"	"
Non-breaking space		

Table 2.1 shows some of the character entities that have to be used for special XHTML characters. The character entities for the < character and the > characters have to be used if you wish to display HTML source code.

As these character entities are considered plain text they can be used as attribute values and in other places that prohibit XHTML mark-up.

2.8 Exercises

In the following exercises install all the pages on your own server.

Ex. 2.30: Create a home page, describing yourself and your interests. Remember create the page using a text editor.

Ex. 2.31: Create a page describing your favourite hobby. Include links to similar pages.

Ex. 2.32: On your hobby page add a table of contents at the top, with links to named anchors within the page.

Ex. 2.33: Experiment with creating HTML links to external pages and to named anchors within the external pages.

Ex. 2.34: Use the list elements to construct a page containing lists of your favourite Cds, Books, Films

Ex. 2.35: Experiment with incorporating images into pages. Create an image intensive page. Load the page with auto-image load turned off on your browser. Where does the `ALT` text appear?

Ex. 2.36: Create an image intensive page with text. Do not include the `WIDTH` and `HEIGHT` attributes of the `IMG` element.

Now add the `WIDTH` and `HEIGHT` attributes of the images. Reload the page after clearing the browser memory and cache of the images so that they will be downloaded again. Is there any difference?

Ex. 2.37: What happens if the `WIDTH` and `HEIGHT` attributes of the `IMG` element are different to that of the loaded image?

2.9 Questions

Short Answer Questions

Q. 2.38: Why should *logical* text elements be used in preference to *font* text elements.

Q. 2.39: What is an *attribute*?

Q. 2.40: Why should embedded images always include the `ALT` attribute?

Q. 2.41: Why is it that some end tags are optional?

Q. 2.42: What is the difference between an XHTML tag and an XHTML element?

Q. 2.43: What is a *name* anchor? How is it defined?

Q. 2.44: Explain how 6 hexadecimal numbers can describe a colour?

Q. 2.45: What is the difference between "block level elements" and "text level elements".

Q. 2.46: Why should embedded images always include the `WIDTH` and `HEIGHT` attributes?

Q. 2.47: Why can frames cause navigation problems?

Q. 2.48: Write a basic frameset document.

Q. 2.49: Write a basic 2x3 table.

Q. 2.50: Why are frames given "names"?

Q. 2.51: What is the rendering difference between the `TD` element and the `TH` element?

Q. 2.52: How are browser windows "targetted"?

2.10 Further Reading and References

- The examples directory for this module on the course web site.
- The World Wide Web consortium HTML4.0 definition can be found in the course resources directory.
- The World Wide Web consortium XHTML definition can be found in the course resources directory.
- The World Wide Web consortium has all the specifications for anything related to the web. Their web site is at

  http://www.w3c.org/.

  The site contains many links for HTML specifications and tutorials.

Department of Mathematics and Computing, USQ.
(This file created: June 12, 2012)

Chapter 3 Cascading Style Sheets

It soon became apparent in the infancy of the Web that the default rendering of HTML was too restrictive. New elements and style attributes were added to the language without control—browsers only recognised the additions they had added to the language. HTML was growing uncontrollably and becoming cumbersome. To address these problems and to try and split style from content—the goal of the original HTML—a "style language", independent of HTML was developed—Cascading Style Sheets (CSS).

The version of CSS we will discuss below is CSS2.

Examples, of using CSS to style XHTML documents can be found in the examples directory for this module on the course web site.

Chapter contents

3.1 Content and Style

A popular term used these days is "multi-tasking"—many people claim they can multi-task—that is, perform multiple tasks simultaneously!

In fact, very few people have (if anyone has) the ability to multi-task—what actually happens is that people task-switch—switch from one task to another—concentrating exclusively on the current task. If

multi-tasking was possible then it would not be illegal to drive and talk on a mobile phone.

Task-switching is tiring and leads to errors—in all tasks. When writing a document concentrating on content simultaneously with formatting invariably leads to errors. After all, producing content itself requires concentrating on the topic, grammar and spelling—formatting is an unnecessary burden—and should be done before starting the document or section or after completing the document or section.

Originally HTML and the web were developed to allow large groups of physicists to collaborate easily over the Internet. The idea was to develop a markup language that would allow an author to concentrate on content and not concentrate on the visual layout.

To this end HTML tried to be a *logical* formatting language. By that, we mean the language contains no reference to the physical media used to display the formatted document. For instance, the HTML element `H1` is used to format a major heading. As the author concentrating on content you do not need to have control over the font being used, the style of the font, size of the font, the colour of the text, the colour of the border, etc. All you need to know is that this markup element will display something that will be recognisable as a header.

Unfortunately, this did not take into account an author's not unreasonable desire, to have control of the final output. What happened was that more and more attributes were added to elements, attributes that controlled the look of the rendered markup. This was done in such a haphazard way that the language has become overburdened with attributes. The addition of more style "attributes" means that content and style become two tasks that are performed simultaneously—because they occur in the same language.

The problems of content and style where recognised in HTML—so a style language "Cascading Style Sheets" (CSS) was developed—completely different from HTML but used to style the "logical" HTML elements.

3.2 Accessibility

As a web document author you have no idea what hardware or software a visitor to your site may be using. In the discussion above, it was pointed out that as author you do not have control over the font used in the HTML element `H1` — this presupposes that the visitor to your site is using a browser were the concept of a "font" has meaning! Braille or Aural browsers for the blind do not have a concept of a "font" — but do understand the idea of a "major heading".

The number of devices that can access a web page or the operating contexts that a visitor to your site may be in can be very different from your own. For example users may:

- not be able to see, hear, move or be able to process some types of information at all.

- may not be able to use or have access to a keyboard or mouse.
- have a text-only screen, small screen (e.g. mobile phone), or slow connection (e.g. modems, wireless, mobile phone).
- be in situations where there eyes, ears or hands are busy or interfered with (e.g. loud environment, busy doing some other task).
- have a different browser, an earlier version of a browser, a voice browser, or a different operating system.

Logical markup is the obvious markup style that should be used for web documents since the author cannot anticipate the user's environment. Logical markup makes as few assumptions about the clients environment as possible—allowing the client's web browser make the display decisions.

Before the advent of CSS many of the style additions to the HTML language were "physical markup" elements and attributes. For example, the `FONT` element, hard units such as *points*, *pixels*, *centimetres* etc. The problem raised by physical markup soon became apparent — sites were (and are) being designed specifically for the hardware owned and used by the designer! Little thought or consideration was (or is) given to people with differing needs or resources.

Example 3.1: How often have you gone to a site and seen the words—

> Best viewed using Internet Explorer X or better, Firefox 3.x or better
>
> or
>
> Best viewed at 1280x1024 resolution.
>
> At best these sites show that the author has a woeful knowledge of how to use the standards to create web documents that work in all environments. At worst it shows a contempt for the web and the client.
>
> Properly authored web documents should not require such statements as they will look "reasonable" on any hardware or using any software.

Example 3.2: A basic difference between client hardware is the resolution of the video card. When designing web pages NO assumptions should be made about the resolution of the clients screen. For example, my video card has a resolution of 1920×1080 at 95 dots per inch. Web pages with font size definitions designed for video cards with a smaller resolution are unreadable on the higher resolution video card.

Some other differences that should be considered when designing web pages are:

- Internet connection speeds
- Video card memory which controls screen resolution and colour depth

- Software used. Web standards are more likely to display correctly across platforms and browsers.

The web should be a medium that makes "information" accessible to everyone. With the introduction of "physical markup" the universality of the web was lost. By introducing a style language separate from content—the content markup language (X)HTML again becomes a "logical" markup language—that can be interpreted by any browser, on any hardware, for any user, in any situation.

Cascading style sheets are a powerful and flexible way of specifying formatting information for web pages. They let you define the font style, font size, background and foreground colours, background image, margins, and other characteristics for *each of the standard logical XHTML elements*. Style sheets allow the author to customise the look and feel of each XHTML element. This means that XHTML elements truly become "logical elements" that describe the conceptual modifications required to the document without any reference to a "physical change". The physical changes have all been incorporated into a style sheet definition. Style sheets can then describe the physical modifications required for differing media types — but the XHTML markup of the document does not change.

Another advantage of using style sheets is that page design is divorced from page content. A page author can concentrate on the page content using standard XHTML logical markup. The *look and feel* of the site can be a separate issue dealt with by using style sheets.

The formatting rules of style sheets are applied in a hierarchical or "cascading" manner. This lets the default rules from the site style sheet combine with any special design rules required for the individual pages which can then be combined with explicit rules from the client. This allows all interested parties, the site designer, the page author and the client have some say in the final outcome.

Exercise 3.3: To see the effect of style sheets, visit the course resource site

`http://www.sci.usq.edu.au/courses/CSC2406/semester2/`

with style sheets turned off in your browser (If you can — not all browsers have this option. If your browser does not, download the home page and view it in your browser from your local disk with the style sheet `LINK` removed from the document's header).

Style sheets can be loaded via URLs, permitting the sharing of style sheets and letting authors change the look and feel of an entire web site by changing only a single style sheet file.

The current CSS definitions are defined as *Level 1 Level 2* and *Level 3*. All browsers have implemented the Level 2 definition which supersedes the Level 1 definition. This section's description is applicable to Level 2 style sheets only.

Figure 3.1: One possible rendering of the screen style of Example 3.4.

Figure 3.2: One possible rendering of the print style of Example 3.4.

3.3 Including Style Commands in (X)HTML

There are a number of ways to incorporate style sheets into an HTML document:

- explicitly code the style sheet information inside the `HEAD` container of the document, using the HTML `STYLE` element.
- import remote style rules from a style sheet document
- specify style information directly in the body of the document.
- or a combination of any of the above.

3.3.1 STYLE Element

Attributes: `TYPE` (required), `MEDIA`

This element must appear within the `HEAD` element of the HTML document. It contains the CSS formatting rules for the document. For example:

Example 3.4: An example of using the `style` element.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
      "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
 <head>
   <title>CSS: Font Families</title>
   <style type="text/css" media="screen">
 <!--
  h1  {font-family: helvetica, sans-serif;
       color: maroon;
       text-align: center;
       text-decoration: underline;}
 -->
```

```
  </style>
  <style type="text/css" media="print">
<!--
 h1  {font-family: "new century schoolbook", serif;
      text-align: center;}
-->
 </style>

</head>
<body>
<h1>Header Aligned: Center</h1>
</body>
</html>
```

Figure 3.1 shows the rendered page for the screen and Figure 3.2 shows the rendered page for printing.

Note

The `style` element should only contain CSS commands not XHTML commands. There is one exception however, to ensure that CSS commands are not displayed by a browser that does not understand your style commands they should be surrounded by XML comment tokens `<!-- ... -->`. The CSS parser ignores them and the XHTML parser ignores any text between them.

`TYPE` This attribute defines the style sheet language. There is no default language, so it must always be defined. The standard language, and the language described in this module is "*text/css*".

`MEDIA` This attribute specifies the intended display medium that the style rules are designed to apply. The attribute value can be a comma separated list. Some of the currently defined media values are:

- `screen` for computer screens;
- `print` for printed documents;
- `braille` for braille, tactile devices;
- `aural` for speech synthesisers;
- `all` all devices;

Exercise 3.5: Copy Example 3.4 to a file and change the media type in the example above to `print`. Load the file into a web browser using the `file://` URL. How is it formatted? Have a look at the document using "print preview" option of your web browser. How is the print version formatted?

The default value is `all`. With the `MEDIA` attribute the document author can specify different style rules for different display media. For example, the rules for print media are normally very different from the rules for screen media. Style for print media would normally have navigation links removed, background images removed, monochrome instead of colour, different fonts, font sizes specified in printers points, etc.

Figure 3.3: One possible rendering of the screen style of Example 3.6.

3.3.2 External Style Sheets

If a style sheet is to be designed to give a site the same look and feel then the style sheet rules should be placed in a separate document. The style sheet document can be linked to the site pages by using the XHTML `LINK` element.

For example:

```
<link rel="stylesheet"
      title="Site Style"
      href="/site-style.css"
      type="text/css"
      media="screen" />
```

`REL` The relationship attribute specifies that the link is to the main `STYLESHEET`. An alternate style sheet can be specified with the value `ALTERNATE STYLESHEET` and setting the `TITLE` attribute. The web browser should supply some mechanism to switch between style sheets.

`TITLE` The title of the style sheet. If this attribute is set then this style sheet is assumed to be the authors preferred style sheet.

`HREF` The URL of the style sheet.

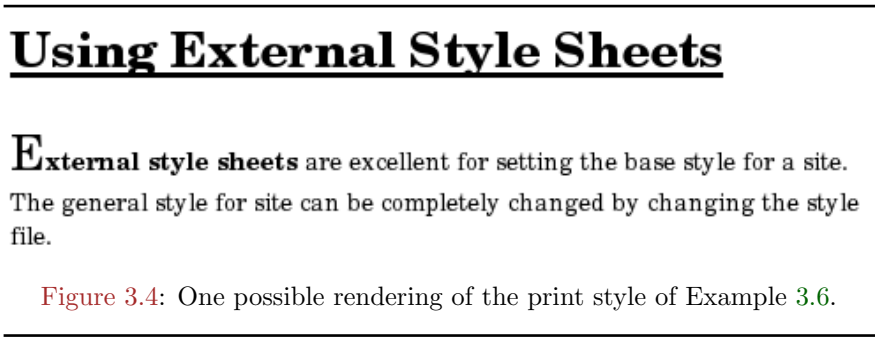
`TYPE` The style language used in the style sheet.

`MEDIA` The display media the style sheet is intended for. This attribute is optional—the default action is to apply it to all media.

If multiple style sheet links appear in the document `HEAD` then the styles are blended in the order they appear provided they have the same `TITLE` (or no title).

Example 3.6: An example of using external style sheets.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
      "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
 <head>
   <title>CSS: External Style Sheets</title>
```



Using External Style Sheets

External style sheets are excellent for setting the base style for a site. The general style for site can be completely changed by changing the style file.

Figure 3.4: One possible rendering of the print style of Example 3.6.

```
   <link rel="stylesheet"        type="text/css"
         href="site_screen.css"  media="screen" />
   <link rel="stylesheet"        type="text/css"
         href="site_print.css"   media="print" />
</head>
<body>
<h1>Using External Style Sheets</h1>
<p>
<strong>External style sheets</strong> are excellent for
setting the base style for a site. The general style for
site can be completely changed by changing the style file.
</p>
</body>
</html>
```

where `site_screen.css` is given by

```
/*
** External style sheet for the Screen
*/

body { color: #000000;
       background-color: #B9B4E5;}

h1, h2, h3 { font-family: Helvetica, san-serif;
             color: #3A0761;}

p:first-letter { font-size: xx-large;
                 font-weight: normal }
```

and `site_print.css` is given by

```
/*
** External style sheet for Printing
*/

body { color: #000000;
       background-color: #FFFFFF;}

h1, h2, h3 { font-family: serif;}

h1 {text-decoration: underline;}

p:first-letter { font-size: xx-large;
                 font-weight: normal }
```

Figure 3.3 shows the rendered page for the screen and Figure 3.4 shows the rendered page for printing.

Exercise 3.7: Go to the course resource web site and experiment with the style sheets of Example 3.6

Exercise 3.8: To download a style sheet and study it all you need is the style sheet's URL. Try the URL:

http://www.sci.usq.edu.au/courses/CSC2406/semester2/build/Site.css

3.3.3 Importing Style Sheets

The Style Sheet construct `import` lets the author break style sheets up into logical sections. Each section can be imported separately. For example:

```
<style type="text/css">
<!--
@import URL(http://www.sci.usq.edu.au/css/margins.css);
@import URL(http://www.sci.usq.edu.au/css/tables.css);
@import URL(http://www.sci.usq.edu.au/css/fonts.css);
-->
</style>
```

The import statement must come first within any style sheet.

Note

The `import` statement is **not** an element within HTML, it is part of the Style Sheet language "text/css".

Everything that is contained within the HTML element `STYLE` is not HTML but written in the Style Sheet language defined by the `TYPE` attribute.

3.3.4 Inline Style

The style sheet specification includes a new attribute added to **all** (X)HTML elements. The `STYLE` attribute allows the author to *inline* the style specifications in the element tag (This attribute is part of the HTML4.0 specification, and thus the XHTML specification). For instance:

```
<h1>Warning Message!</h1>
<p style="margin-left: 4em;
          margin-right: 4em;
          font-size: 200%;
          color: red">
You have just violated the code
of practices of this institution!
</p>
```

will modify the text in the paragraph contained within this `P` element only, of the document.

Separate style rules are easier to extend and maintain than inline styles, and should generally be used. Inline style rules reduces the flexibility inherent in style files. Inline styles are useful however, in

modifying a style in a one-off manner—required for one page only. This normally happens when the generic style chosen for an element throughout a site is not quite suitable in one or two instances—inline modifications are better than adding all possible variations in a site style file—as the site style file would soon grow unwieldy

3.4 Specifying Style Rules

HTML elements are customised by the use of *style rules*. An example of a rule to modify the colour and size of all `H1` elements is

```
h1 {color: green;
    font-size: xx-large;
   }
```

This *rule* consists of the *selector* (`h1`), the *property* (`color`) and the *value* (`green`). The property/value pairs of a selector can be grouped between braces and separated by semi-colons. The most basic type of selectors are simply the names of HTML elements. The properties listed inside the braces apply to all occurrences of the element inside the document.

3.4.1 Selectors

Style rules are normally defined by using a command of the following form:

```
selector { property1: value1;
           property2: value2;
           ...
           propertyN: valueN
         }
```

The style rules are applied to the specified selector. The common selector type is an HTML element. HTML elements are not the only selector type. The CSS standard allows a variety of selector types to define formatting rules that only apply in certain situations. The six categories of selectors are:

- HTML elements
- HTML elements in certain contexts
- User defined classes
- User defined IDs
- Pseudo classes
- Pseudo elements.

HTML Elements

Property settings are inherited, that is, elements inherit the properties from parent elements (outer elements) if the property has not been explicitly set. For example, suppose the `H1` element has the `EM` element inside it:

```
<h1>This Headline is <em>important</em>!</h1>
```

If the colour for the H1 element has been set to blue and no colour has been assigned to the EM element it will then *inherit* the colour of the parent element, that is blue.

To set a *default* property for all HTML elements, then the property can be defined for the BODY element, which all elements should inherit from[1].

Elements can be grouped in comma separated lists to allow common styles to be set for multiple HTML elements. For example:

```
h1, h2, h3, h4, h5, h6 { color: olive;
                         font-family: sans-serif }
```

would set all header elements, rather than setting each one individually.

Contextual HTML Elements

Inheriting properties is the default action when styles are defined. Sometimes this is not convenient. For example, consider the following styles:

```
body { color: blue; }
h1   { color: red; }
em   { color: red; }
```

This sets the main body to blue text, top-level headings to red, and emphasised text to red. This means that emphasised text within a top-level heading will not be distinguishable from the rest of the heading. So, a rule can be added to specify that emphasised text be green only when inside top-level headings, as follows:

```
h1 em  { color: green; }
```

That is, the style is only applied when the EM element is a "descendant" of an H1 element. The relationship can be arbitrarily deep. To specify a direct "child" element use the child-selector > as follow:

```
h1 > em  { color: green; }
```

In this case the EM element must be a direct child of an H1 element.

There is also a selector for adjacent siblings—a +. This selector will matches if two elements share the same parent and are adjacent to each other. For example—

```
h1 + h2  { margin-top: -0.5ex; }
```

will reduce the white space at the top of the H2 element only if it directly follows an H1 element—thus reducing the white space between the two header elements.

[1] In some browsers the inheritance of properties, are at best, imperfectly implemented! So be wary.

Author Defined Classes

A greater control is given to the author, by the use of selector classes. A class name can be added to an HTML element. For example, to define an "abstract" paragraph style with indented left and right margins and italic text, the following style could be defined:

```
p.abstract { margin-left: 4em;
             margin-right: 4em;
             font-style: italic
           }
```

This defines the class "abstract" for the `P` element.

To use this class in a HTML document, the class name is used as the value for the HTML attribute `CLASS`. For example:

```
<h1>Smoothed Particle Hydrodynamics</h1>
<p class="abstract">
Smoothed Particle Hydrodynamics (SPH) is a Lagrangian
method that calculates spatial derivatives on a
distribution of interpolation points.
</p>
```

All HTML elements have the attribute `CLASS`.

In the example above the class "abstract" can only be used with the `P` element. Classes can be defined that can be used with any HTML element. For example:

```
.blue { color: blue;
        font-weight: bold
      }
.big { font-size: larger; }
```

defines the classes "blue" and "big" which can be used with any HTML element.

```
<h1 class="blue">The use of the class blue</h1>
The text you are reading is in the default colour
but by using the class blue,
<span class="blue">this text is now blue and bold</span>

This text is <span class="blue big">blue, bold and larger</span>
```

As can be seen from the example above the value of the `CLASS` attribute can contain multiple user defined classes—separated by white space.

Author Defined IDs

An ID is like a class, but can only be applied *once* in a document. It is defined by preceding the name with a `#` and referenced with the HTML attribute `ID`, as follows

```
<head>
  <title>ID example</title>
  <style  type="text/css">
    <!--
     #title {color: red}
```

```
     h2#color {color: green}
    -->
  </style>
</head>
<body>
<h1 id="title">The use of the ID attribute</h1>
...
<h2 id="color">Green header, but only once</h2>
</body>
```

The `ID` has major importance, not so much with CSS, but with uniquely defining elements for dynamic modification using Javascript and/or CGI scripts.

Pseudo-Classes and Pseudo-Elements

As seen above style can be attached to elements based on their name or position in the document. Pseudo-classes and elements are used to be able to select characteristics of the document after it has been formatted or as the client interacts with the document—information not available before the document is loaded.

:link This selector matches unvisited anchor elements

:visited This selector matches visited anchor elements

:hover This selector matches the element that has the cursor hovering over it.

:active This selector matches the element that is currently active—the period between a button press and a button release.

:focus This selector matches the element that has the keyboard focus.

:first-line This selector matches the first line of a paragraph.

:first-letter This selector matches the first character of the first line of a block of text.

This list is incomplete though it contains the most useful pseudo classes and elements. See the CSS2 reference for more information on pseudo classes and elements.

3.4.2 Precedence Rules

There are often multiple style rules that can apply to a particular section of text, and the browser needs to know the order in which to apply them. The rules with the highest precedence are applied *last* so that they replace conflicting values from lower priority rules. The rules for determining the precedence (or "cascading") order are as follows:

(a) Rules marked *important* have the highest priority. To mark a style rule as important the tag "`!important`" is appended. For example;

```
body {color:   white !important;
      background-color: black !important;
     }
```

These declarations are used sparingly, if at all. Reader's rules marked `!important` have precedence over author rules marked `!important` (cf. below).

(b) Authors rules have precedence over reader's rules. Browsers that permit readers to specify style rules to override the defaults, will give higher priority to the author's rules (cf. above)

(c) More specific rules have precedence over less specific rules. In general `ID` attributes have a higher priority to `CLASS` attributes, which have a higher priority to element rules. (see the CSS standard for a more detailed discussion of the specificity of rules.)

```
H1#fore {color: black; }
.fore   {color: blue; }
H1.fore {color: green; }
```

All `<H1 class="fore">...</H1>` contents will be green and the `<H1 id="fore">...</H1>` will be black.

(d) If there is a tie with the specificity of rules then the last rule has precedence.

3.4.3 Property URLs

There are a number of situations where URLs have to be defined in style sheets. For example, when importing different style documents, or when specifying a background image.

When specifying an URL as the value of a property the URL must be contained in a `url(...)` construct. For example:

```
body { background-image: url(pattern.jpg) }
```

The specified URL can be relative to the current document or an absolute URL. The normal character escape rules for URLs apply.

3.4.4 Property Units

Cascading style sheets allow authors to specify sizes and colour in a variety of different units.

Lengths

Lengths can be specified in *relative* units or *absolute* units. Relative units specify a length relative to another length property.

Note

Style sheets that use relative units will more easily scale from one medium to another (screen to printer), and are more likely to display clearly on different screens.

So where ever possible (which is almost everywhere) never use absolute or "physical" units use relative units.

Relative Units

em The width of the letter "m" in the current font.

ex The height of the letter "x" in the current font.

px Pixels—relative to the current viewing device. Pixels should be avoided as the current size of the viewing device is unknown!

Absolute Units

pt Points—used in typesetting. There are 72 points to the inch. One inch is 2.54 cm.

pc Picas—used in typesetting. There are 6 picas per inch. One inch is 2.54 cm.

in Inches. One inch is 2.54 cm.

cm Centimetres.

mm Millimetres.

Wherever possible, the best units to use from the list above are `em` and `ex`—as these are defined relative to the current font-size. For example, white-space around text defined using `em` or `ex` will grow and shrink with the size of the text—making it look good on all monitors

Specifying font sizes in points—the traditional typesetting way—is to be avoided as you do not know the default font size the client has chosen in their browser. Client font size will be chosen based on the clients screen resolution, size of monitor, distance from monitor, and eyesight—all things you as the author know nothing about and cannot control.

Example 3.9: The default font size for Internet Explorer is quite large. As most clients never change the default size—web authors have got into the habit of specifying font-sizes explicitly as 9 or 10 points. Which can become unreadable if you have a large monitor, set at arms length and with the default font-size set to 14 points to avoid eye strain (as mine is).

Percentage Units

Percentage values are always relative to another value, for example a length unit. Each property that allows percentage units also defines what value the percentage value refer to.

Colours

A colour is a either a keyword or a numerical RGB specification. (See Section 4.1 on a more detailed discussion of colours.)

color-name This is one of 17 standard colours listed in Table 4.1

`#rrggbb` Standard six character hexadecimal notation for colour.

#rgb Shorthand three character hexadecimal notation. This is converted to the standard six letter notation. For example: `#0AF` will be converted to `#00AAFF`

rgb(rrr,ggg,bbb) The integer values `rrr`, `ggg`, `bbb` are in the range 0 to 255. For example:

```
h1 { color: rgb(200,0,255); }
```

rgb(rrr%,ggg%,bbb%) The floating values `rrr`, `ggg`, `bbb` are in the range 0.0 to 100.0. For example:

```
h1 { color: rgb(50%,0%,100%); }
```

Exercise 3.10: When designing menus for a web page many authors use images as they are of fixed width—which means they can easily align the menus. Explain why using images is not a good idea. Explain how menus constructed using text can have a known width without knowing anything about the font used in the browser.

3.5 Font Properties

The font properties allow the author to specify several aspects of the rendered font: style, family, size and weight.

The properties that follow are not complete. For a complete list of font properties consult the CSS standard.

font-family This property specifies a prioritised list of type faces. The list of family names is separated with comas. For example:

```
body { font-family:  gill, helvetica, sans-serif; }
```

The font family can be a specific font family, for example `gill` or `helvetica`, or a generic font family, such as `sans-serif`.

The available generic font families are `serif`, `sans-serif`, `cursive`, `fantasy` and `monospaced`.

Generic font families are preferred over specific font families, as the author cannot assume the font families available to the user's browser.

font-style This properties has values `italic`, `normal` and `oblique`, and selects the font face within a font family.

font-variant This property has two values `normal` and `small-caps`

font-weight The weight of the font can be specified with the values `normal`, `bold`, `bolder`, `lighter`.

The CSS also specifies the values `100`, `200`, `300`, `400`, `500`, `600`, `700`, `800`, `900`. Where `normal` is synonymous with `400` and `bold` with `700`.

font-size Font size can specified either absolute or relative:

Absolute Units

Symbolic size Absolute symbolic sizes are `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, or `xx-large`.

Length units Absolute length units. See §3.4.4.

Relative Units

Symbolic size Relative symbolic sizes are `smaller` or `larger`.

Length units Relative length units. See §3.4.4.

Percentage Percentages are calculated relative to the parent element's font size.

Relative units or Symbolic sizes are preferred as they will be based on the default size chosen by the client—something as author, you know nothing about.

Example 3.11: The following is an example of using the `font-family` property to create different `H2` headings

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
      "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
 <head>
   <title>CSS: Font Families</title>
   <style type="text/css">
 <!--
  H2.serif     {font-family: "new century schoolbook", serif;
                color: blue;}
  H2.cursive   {font-family: "zapf chancery", cursive;
                color: blue;}
  H2.sansserif {font-family: helvetica, sans-serif;
                color: blue;}
  H2.mono      {font-family: courier, monospaced;
                color: blue;}
 -->
 </style>
</head>
<body>
<h1>CSS: Font Families</h1>

<h2 class="serif">Serif Fonts</h2>
Serif fonts tend to have a relatively narrow, upright
shape with distinctive serifs across the end of the
letter strokes. Serif font faces are generally considered
to have a classic, formal, businesslike font style that
is reliable for the Web.

<h2 class="sansserif">Sans-Serif Fonts</h2>
Sans serif fonts have a very regular, often geometrical
shape with no serifs on the end of the letter strokes.
Sans-serif font faces are generally considered to have a
simple, clean, modern font style that is reliable for the Web.
```

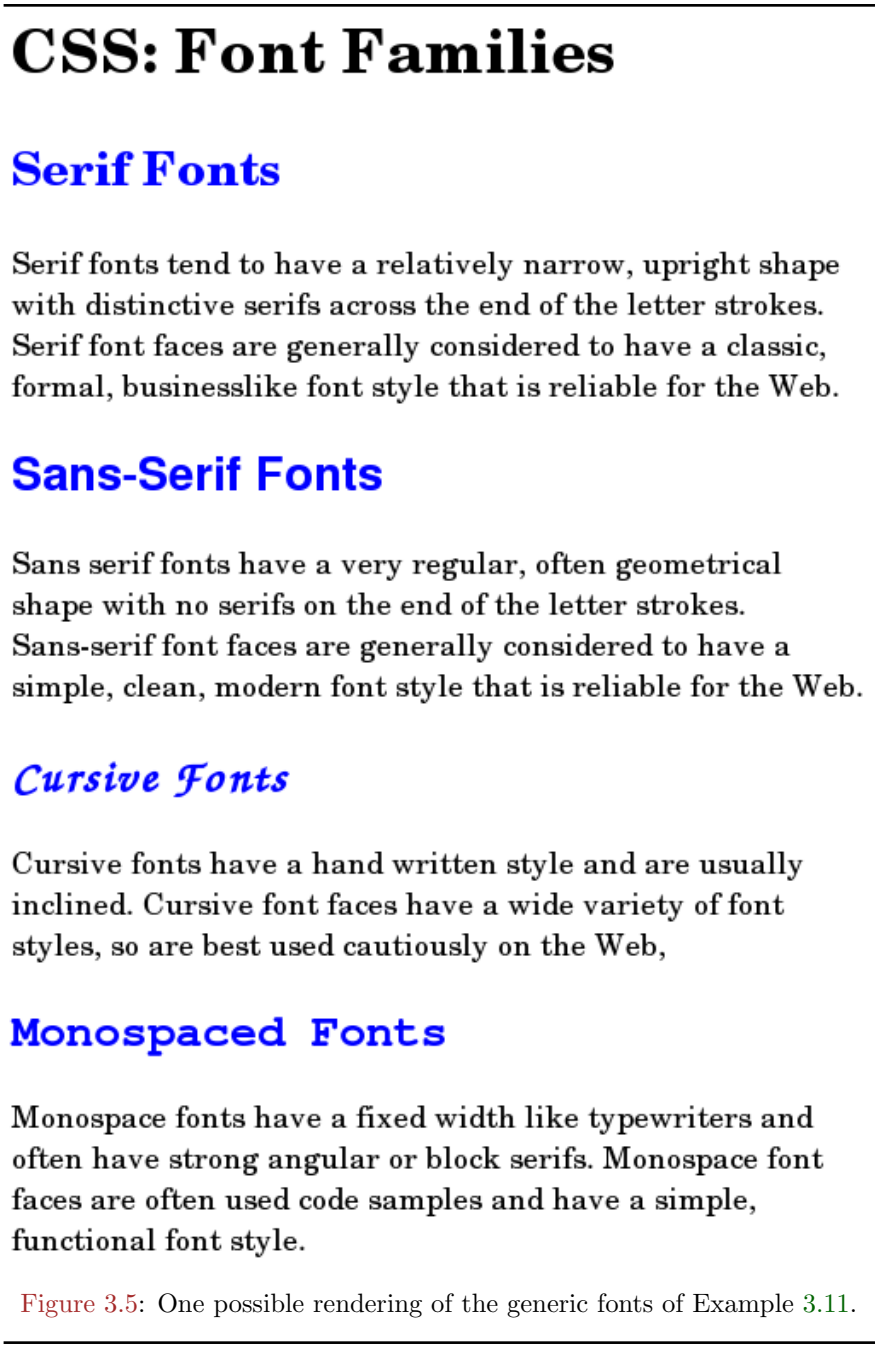


Figure 3.5: One possible rendering of the generic fonts of Example 3.11.

CSS: Font Properties

The SPAN element can be used to change text properties within a line of text. ***For example***, the preceding text become **bold-italic**, by using the *"bolditalic"* class.

Figure 3.6: One possible rendering of the font properties of Example 3.12.

```
<h2 class="cursive">Cursive Fonts</h2>
Cursive fonts have a hand written style and are usually
inclined. Cursive font faces have a wide variety of font
styles, so are best used cautiously on the Web,

<h2 class="mono">Monospaced Fonts</h2>
Monospace fonts have a fixed width like typewriters and
often have strong angular or block serifs. Monospace font
faces are often used code samples and have a simple,
functional font style.
</body>
</html>
```

Figure 3.5 shows the rendered page using the generic font families.

Example 3.12: The following is an example of using font properties with the `SPAN` element.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
      "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
 <head>
   <title>CSS: Font Families</title>
   <style type="text/css">
 <!--
  h1 { font-weight: bold;
       font-family: sans-serif;
       font-size: x-large;
       color: green; }
  samp { font-size: larger; }
  .bold    { font-weight: bold; }
  .italic  { font-style: italic; }
  .bolditalic { font-style: italic;
                font-weight: bold;
                color: maroon;  }
 -->
 </style>
</head>
<body>
<h1>CSS: Font Properties</h1>
The <samp>SPAN</samp> element can be used to change text
properties within a line of text.
```

```
<span class="bolditalic">For example</span>, the preceding
text become <span class="bold">bold-italic</span>,
by using the <span class="italic">&ldquo;bolditalic&rdquo;</span>
class.
</body>
</html>
```

Figure 3.6 shows one possible rendering of the font properties used in this example.

3.6 Foreground and Background Properties

Cascading style sheets support changing the foreground colour, background colour, and background images for regions of text. If foreground colours are to be changed then background colours should also be changed. The author cannot assume the default colours of the user, if text is not to disappear into the background then the author of the style sheet should explicitly define all colours.

Note

Do not assume that the default background colour of all browsers is white! The user can change the default background colour of the browser—I set it it to a light-grey or off-wite as they are less fatiguing colours than a brilliant white.

If you assume white then set it in your style sheet.

The properties that follow are not complete. For a complete list of foreground and background properties consult the CSS standard.

color Specify the foreground, or text colour of the element.

background-color Specify the background colour of the element. If the keyword `transparent` is used then an inherited colour will show through.

background-image This specifies the image to use as the background of the element. Authors should always supply a background colour to use if the image is unavailable or the user has disabled image loading.

background-repeat This property specified how the image is to be tiled. The possible values are:

- `repeat` Tile the image in both directions
- `no-repeat` Display the image once only. Do not tile.
- `repeat-x` Tile the image in the x-direction only.
- `repeat-y` Tile the image in the y-direction only.

background-attachment This property determines if the background image scrolls with the content (value: `scroll`, the default action) or is fixed (value: `fixed`).

background-position This property specifies the initial position of the image with respect to the upper left hand corner of the element it is associated with. The position is specified as a pair of values. The first is the horizontal position the second is the vertical position. Allowable horizontal values are `left`, `center`, `right`, a percentage distance,

or a distance in standard units. Allowable vertical values are `top`, `middle`, `bottom`, a percentage distance, or a distance in standard units. In percentage units `0% 0%` is the top-left corner, and `100% 100%` is the bottom-right.

Negative positions are permitted, allowing images to hang into margins or previous text sections.

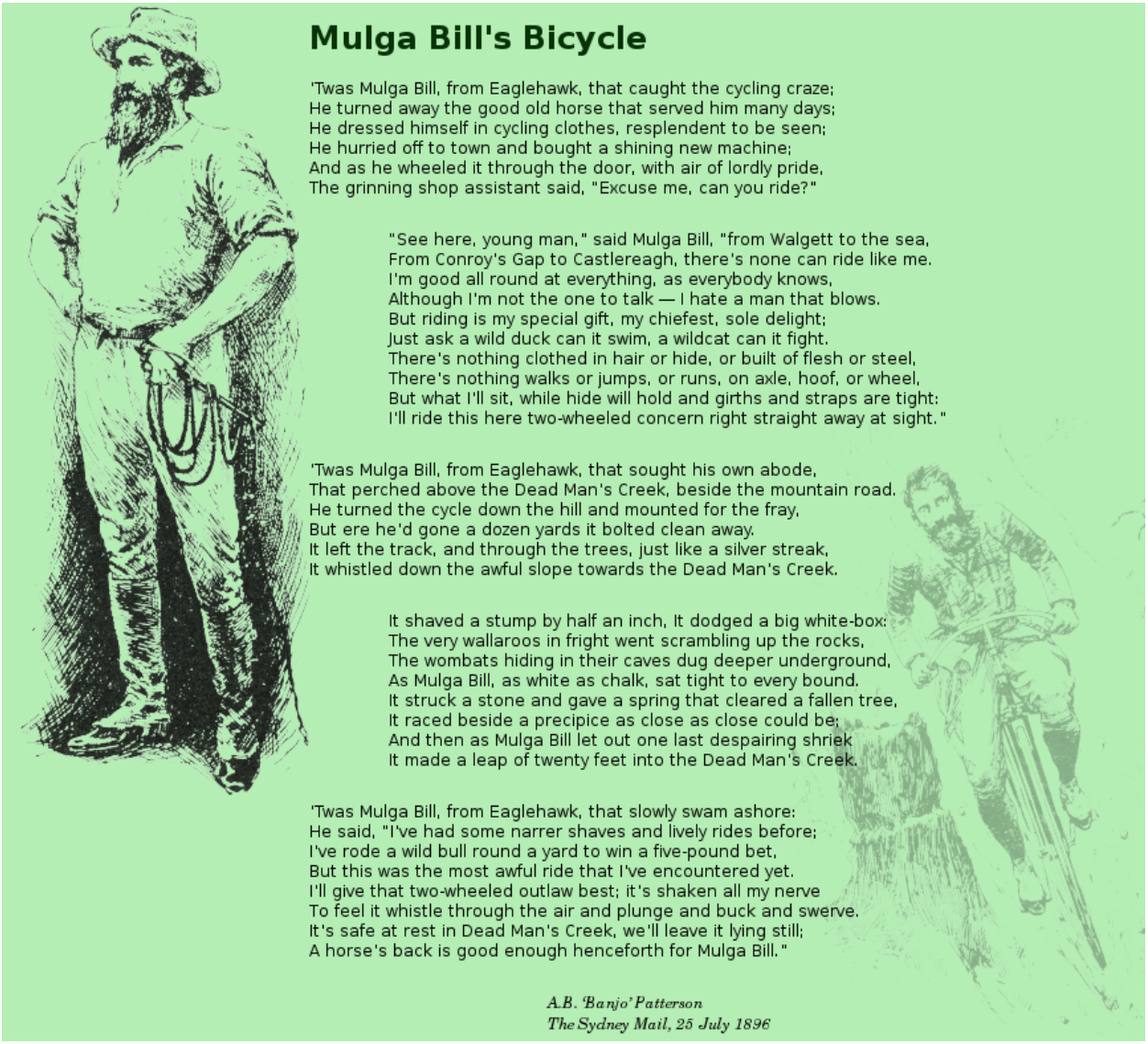


Figure 3.7: One possible rendering of Example 3.13.

Example 3.13: An example using background images.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
      "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
 <head>
   <title>Mulga Bill's Bicycle</title>
   <style type="text/css">
<!--
    body  { color: black;
            background: #b4eeb4;
            margin-left: 235px;
            background-image: url(mbill1f.png);
            background-repeat: no-repeat;
            background-attachment: fixed;}
    h1#heading { font-family: sans-serif;
                 font-size: xx-large;
                 font- weight: bold;
                 color: #073107; }
    div#poem { background-image: url(mbill2f.png);
               background-repeat: no-repeat;
               background-attachment: scroll;
               background-position: bottom right;}
    div#poem pre { font-family: sans-serif;
                   margin-bottom: 2em; }
    div#poem pre.even { margin-left: 5em; }
    div#poem pre.odd  { margin-left: 0pt; }
    div#poem div.author { margin-left: 15em;
                          font-style: italic;}
-->
    </style>
 </head>
 <body>
 <h1 id="heading">Mulga Bill's Bicycle</h1>
 <div id="poem">

 <pre class="odd">
&rsquo;Twas Mulga Bill, from Eaglehawk, that caught the cycling craze;
He turned away the good old horse that served him many days;
He dressed himself in cycling clothes, resplendent to be seen;
He hurried off to town and bought a shining new machine;
And as he wheeled it through the door, with air of lordly pride,
The grinning shop assistant said, &ldquo;Excuse me, can you ride?&rdquo;
</pre>
<!--Full text of HTML page available from course web site-->
 <div class="author">A.B. &lsquo;Banjo&rsquo; Patterson
               <br />The Sydney Mail, 25 July 1896
 </div>
</div>
</body>
</html>
```

Figure 3.7 shows the rendered page.

3.7 Text Properties

The text properties control the way text in a paragraph is laid out. This applies not just to the text in a paragraph but also to the inline-elements found in the paragraph—in particular the `IMG` element.

The properties that follow are not complete. For a complete list of text properties consult the CSS standard.

word-spacing This property specifies a change to the default spacing between word. Values are expressed using the standard length units. Positive values indicate an increase, negative values indicate a decrease.

letter-spacing This property specifies a change to the default spacing between characters. Values are expressed using the standard length units. Positive values indicate an increase, negative values indicate a decrease.

text-decoration As the name suggests, this property specifies text "decorations". That is, `none`, `underline`, `overline`, `line-through`, and `blink` (this should never be used as it is too distracting on a page). An example is to make hypertext-text links blue, without an underline:

```
:link { color: blue; text-decoration: none }
```

vertical-align This property determines how elements are positioned vertically. The value indicates how far to raise or lower the baseline of the element, relative to the parent element. A percentage value is relative to the line height of the element itself. Allowed symbolic values are:

- **baseline** Align the baseline of the element with its parent.
- **middle** Align the midpoint of the element with the baseline plus half the x-height of the parent (experiment with this one).
- **sub** Subscript the element.
- **super** Superscript the element
- **text-top** Align the top of the element with the top of the parent element's font.
- **text-bottom** Align the bottom of the element with the bottom of the parent element's font.
- **top** Align the top of the element with the tallest element on the line
- **bottom** Align the bottom of the element with the lowest element on the line

text-transform Transform the text to all uppercase (`uppercase`), to lowercase (`lowercase`), have the first letter of each word uppercase (`capitalize`), or neutralise inherited transforms (`none`).

text-align This property describes how text is aligned within the element. The possible values are `left`, `right`, `center`, and `justify`.

text-indent This property specifies the indentation of the *first* line of a paragraph. It is calculated with respect to the existing left margin as specified by `margin-left` (see below). Values can be normal length units or can be percentages relative to the parent elements width.

line-height This property specifies the height of each line. This is the distance between two consecutive baselines in a paragraph. In addition to the standard length units, a percentage value can be supplied, interpreted with respect to the font size.

Example 3.14: The XHTML element `IMG` is by default an inline element. It is used to embed and image into a line of text. How the embedded image is to align with the text can be configured using CSS text properties for vertical alignment and the `float` property to allow text to flow around the image. Figure 3.8 shows the different alignment effects.

3.8 Bounding Box Properties

Cascading style sheets assume that all elements will result in one or more *rectangular* regions. This region is known as the "bounding box" and contains a margin, border, padding area and the contents of the box, each nested within the other. Figure 3.9 illustrates the bounding box.

The properties that follow are not complete. For a complete list of bounding box properties consult the CSS standard.

The width and height of the total box is the sum of the width and heights of the main element, the padding that surrounds it, the border surrounding the padding, and the margins surrounding the border. Margins are always transparent, letting the colour and/or image underneath to show through. The padding always takes on the background colour or image of the element. The border, can have its own background.

Note The default value for the properties width, height and margin are 'auto'. That is, the browser will automatically assign these values based on box contents and floating rules.

How all the rules interact to prodcue the final values can be complex and confusing—if a box is not bahving in the way you expect you need to read carefully the CSS reference on computing height, width and margins.

Note Though CSS defines margins for every element, how margins interact is not at all clear. As a rough rule of thumb, horizontal margins are always counted when placing boxes side-by-side, vertical margins collapse—only the widest margin of two vertically stacked boxes is used. Unless the box has been "floated" or "absolutely" positioned (see Section 3.9 below) then margins do not "collapse".

Example of aligning elements

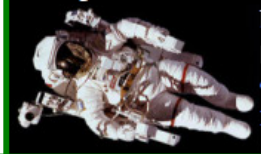
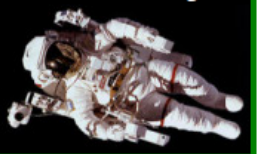
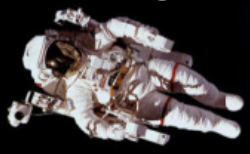
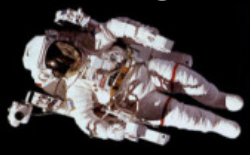
Alignment	Result
LEFT	The `float:left` style floats the image to the left side, with text flowing around it on the right. The space around the image can  be adjusted using the `margin` or `padding` style properties of the image element. Any element can use the `float` property.
RIGHT	The `float:right` style floats the image to the right side, with text flowing around it on the left. The space around the image can be adjusted using the `margin` or `padding` style properties. Any element can use the `float` property. 
TOP	The `vertical-align:top` style property positions the image  so that the top of the line of text containing the image is aligned with the top of the image.
MIDDLE	The `vertical-align:middle` style property positions the image  so that the baseline of the text containing the image is aligned with the middle of the image.
BOTTOM	The `vertical-align:bottom` style property positions the image  so that the baseline of the text containing the image is aligned with the bottom of the image.

Figure 3.8: Using style commands to align an inline image with the surrounding text.

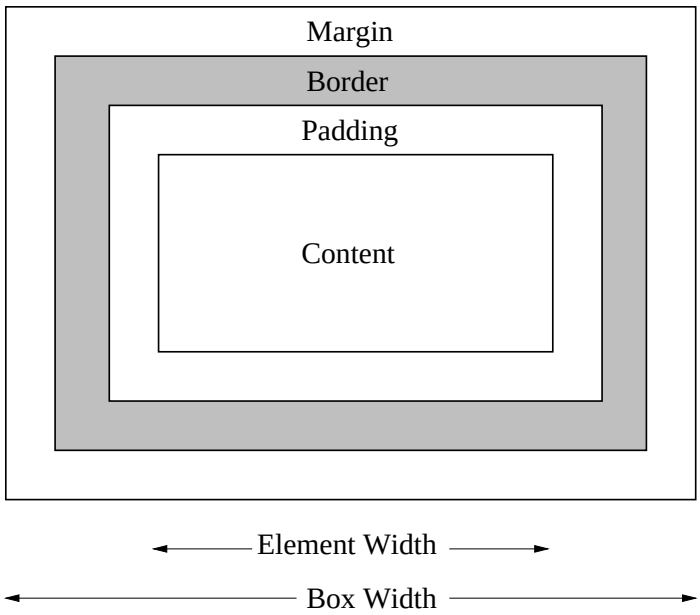


Figure 3.9: Illustration of an element bounding box

See the CSS reference for a detailed and confusing description of collapsing margins.

width This property sets the width of the bounding box overriding the default width which is determined from the contents. The width is specified using normal length units, percentages or the keyword `auto`.

height This property sets the height of the bounding box overriding the default height which is determined from the contents. The height is specified using normal length units, percentages or the keyword `auto`.

margin-left This property sets the left margin using normal length units, percentages or the keyword `auto`. Negative values are permitted; though the result is implementation specific.

margin-right This property sets the right margin using normal length units, percentages or the keyword `auto`. Negative values are permitted; though the result is implementation specific.

margin-top This property sets the top margin using normal length units, percentages or the keyword `auto`. Negative values are permitted; though the result is implementation specific.

margin-bottom This property sets the bottom margin using normal length units, percentages or the keyword `auto`. Negative values are permitted; though the result is implementation specific.

border-left-width This property sets the width of an element's left border, using normal length units, or the keywords `thin`, `medium`, or `thick`. Border widths cannot be negative.

border-right-width This property sets the width of an element's right border, using normal length units, or the keywords `thin`, `medium`, or `thick`. Border widths cannot be negative.

border-top-width This property sets the width of an element's top border, using normal length units, or the keywords `thin`, `medium`, or `thick`. Border widths cannot be negative.

border-bottom-width This property sets the width of an element's bottom border, using normal length units, or the keywords `thin`, `medium`, or `thick`. Border widths cannot be negative.

border-color This sets the border colours. One to four values can be supplied, specifying the colour of the top, right, bottom, and left borders. If only one colour is supplied it applies to all four borders. If two or three colours are supplied, colours for any missing border are taken from the border directly opposite.

border-style This property specifies the way in which the borders will be drawn. One to four values are supplied, specifying characteristics for the top, right, bottom, and left borders in the same manner as `border-color`. Each value can be one of `none`, `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset`, and `outset`.

padding-left This property sets the width of the padding on the left of the element, using normal length units, or percentages. Percentages are interpreted with respect to the parent element's width and height.

padding-right This property sets the width of the padding on the right of the element, using normal length units, or percentages. Percentages are interpreted with respect to the parent element's width and height.

padding-top This property sets the width of the padding on the top of the element, using normal length units, or percentages. Percentages are interpreted with respect to the parent element's width and height.

padding-bottom This property sets the width of the padding on the bottom of the element, using normal length units, or percentages. Percentages are interpreted with respect to the parent element's width and height.

Example 3.15: An example of using the bounding-box to modify an `H1` header.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
      "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
 <head>
   <title>CSS: STYLE example</title>
   <style type="text/css" media="screen">
 <!--
```

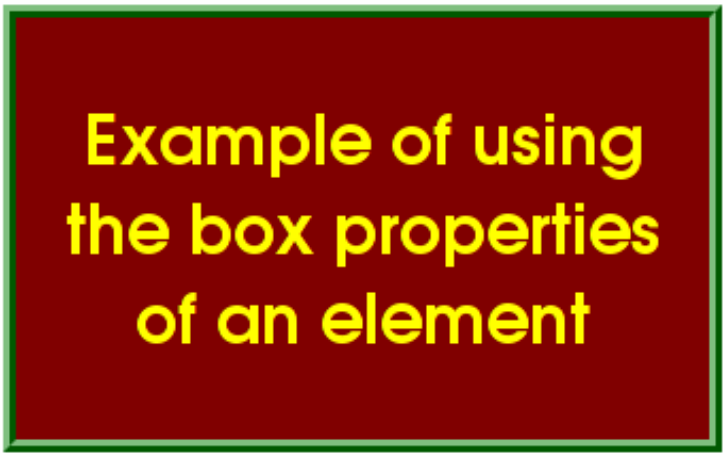


Figure 3.10: One possible rendering of Example 3.15.

```
  body { margin:  0px;
         padding: 0px;
        }
  h1.boxed { font-weight: bold;
             font-family: sans-serif;
             font-size: xx-large;
             text-align: center;
             color: yellow;
             background-color: maroon;
             padding-top:    2ex;
             padding-bottom: 2ex;
             padding-left:   2.5%;
             padding-right:  2.5%;
             width: 50%;
             margin-left: 22%;
             margin-right: 22%;
             margin-top: 4ex;
             margin-bottom: 4ex;
             border-color: green;
             border-style: ridge;
             border-left-width: 6px;
             border-top-width: 6px;
             border-bottom-width: 6px;
             border-right-width: 6px;
     }
 -->
   </style>

</head>
<body>
<h1 class="boxed">Example of using the box properties of an
element</h1>
```

```
</body>
</html>
```

Figure 3.10 shows the rendered header.

Exercise 3.16: Consider the CSS for Example 3.15 above. Explain why the horizontal percentages do not add to 100%—but the heading looks centred (on my browser at least). Why?

Is the heading still centred (if it was) if the browser's horizontal width is changed?

Exercise 3.17: Example 3.15 gives one method of centering a box horizontally—by having the box witdth, padding and margin add to 100%.

But with the inclusion of a fixed border width it starts to become problematic. One way to overcome this problim is to get the rendering engine to calculate the widths for you—this is done by specifying the value `auto` on the left and right margin.

Rewrite Example 3.15 using the value `auto` (instead of percentages) for left/right margins, padding and width. Be systematic only make one change at a time. What are the results?

Example 3.18: The Beowulf translation example from the XHTML module has been modified using style properties. With style sheets turned off, the new rendered document is identical to the old vanilla XHTML rendering, with style sheets turned on, the page is considerably changed.

3.9 Box Positioning Properties

In CSS many box positions and sizes are calculated with respect to the edges of a rectangular box called a `containing block`. In general, generated boxes act as containing blocks for descendant boxes; that is a box defines the containing block for its descendants.

Each box is given a position with respect to its containing block, but it is not confined by this containing block; it may overflow it. The `BODY` element generates a box that serves as the initial containing block for subsequent layout.

When an HTML page is constructed—boxes are laid out one after another, vertically beginning at the top of the containing block with each box's left outer edge touching the left edge of the containing block. The boxes are laid out as they are encountered in the HTML document–this is called the "normal flow". The *normal flow* of the document can be modified by moving boxes around the page.

Apart from "normal flow" CSS has two other positioning schemes: Float and Absolute positioning.

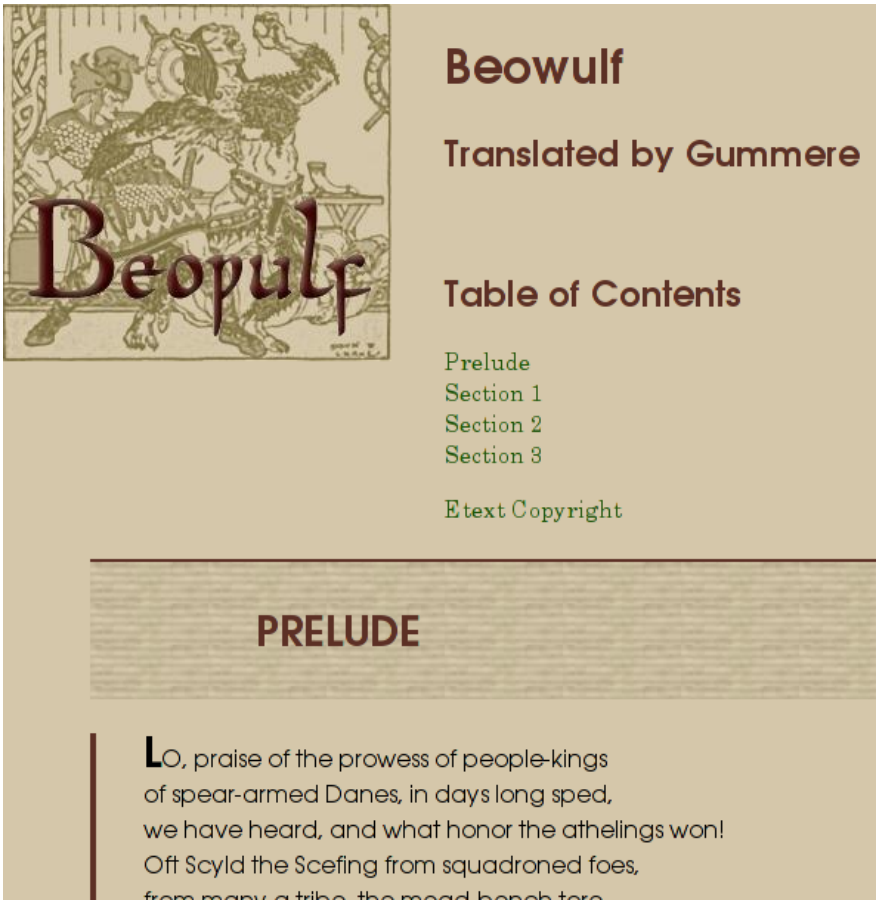


Figure 3.11: Addition of style commands to Beowulf Translation originally used in the XHTML module 2.

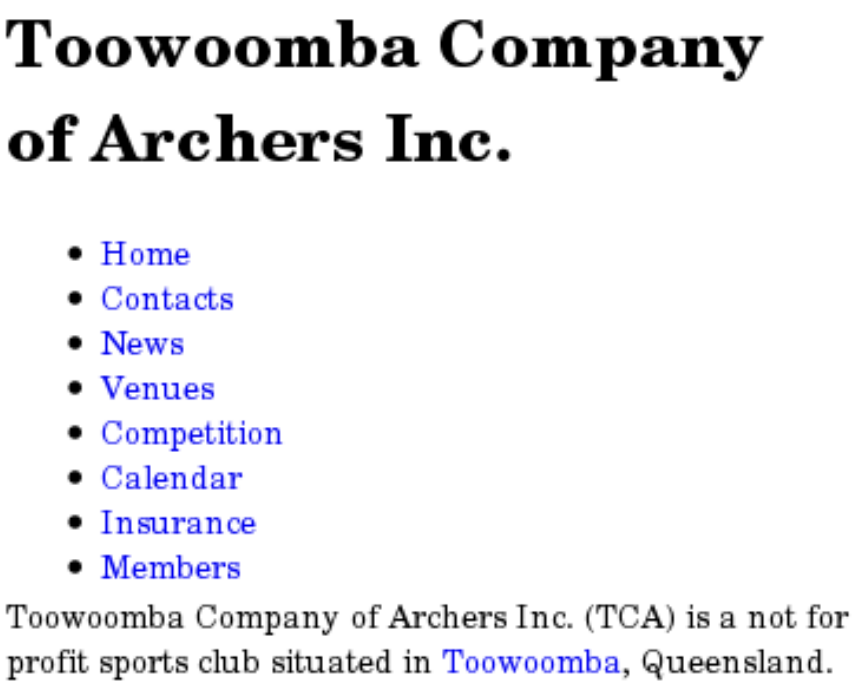


Figure 3.12: The home page for the Toowoomba Company of Archers Inc. without style commands.

Floating Positioning Scheme

A box is first laid out according to the normal flow, then taken out of the flow and shifted to the left or right as far as possible. Content may flow along the side of a float.

float This property specifies whether a box should float to the left , right, or not at all—with values: `left`, `right` or `none` respectively.

clear This property controls flow around a floated box. It indicates which sides of an element's box may *not* be adjacent to an earlier floated box. If the value is—

- `left` then this box will be pushed below any left-floated boxes.
- `right` then this box will be pushed below any right-floated boxes.
- `both` then this box will be pushed below any floated boxes.
- `none` there is no constraint on the box's position with respect to floats.

Example 3.19: In the course examples directory for this module can be found the welcome page for the Toowoomba Company of Archers Inc. web site. The page uses CSS commands for all formatting.

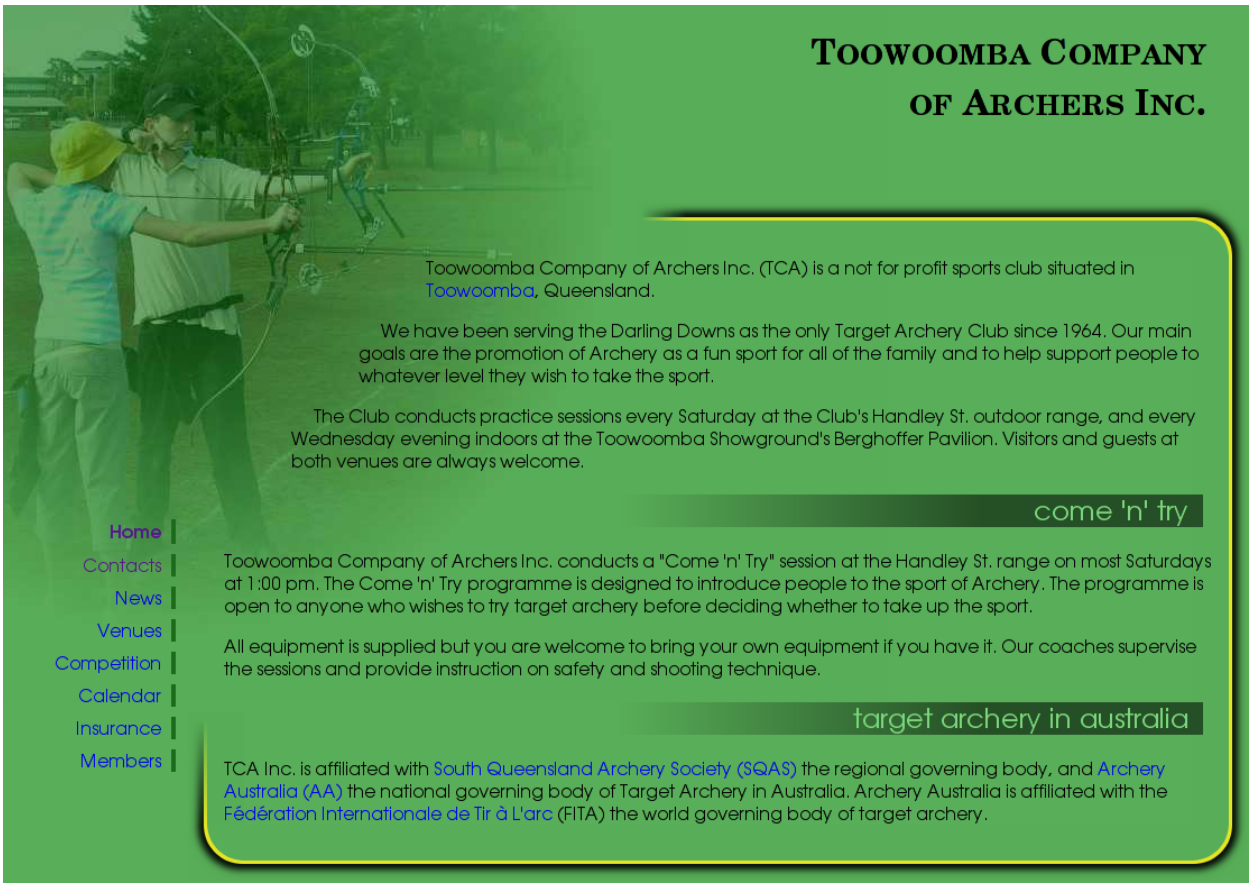


Figure 3.13: The home page for the Toowoomba Company of Archers Inc. with style sheets.

```
<!DOCTYPE HTML PUBLIC
   "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xml:lang="en"
      xmlns="http://www.w3.org/1999/xhtml"
      lang="en">

<head>
    <title>Toowoomba Company of Archers Inc.</title>
    <link rel="stylesheet"
          href="TCA.css"
          title="Site Style Sheet"
          type="text/css">
</head>

<body>
 <div class="header">
   <h1>Toowoomba Company<br> of Archers Inc.</h1>
 </div>

 <div class="menu">
 <ul>
   <li class="current">
      <a href="index.html">Home</a></li>
   <li><a href="contact.html">Contacts</a></li>
   <li><a href="news.html">News</a></li>
   <li><a href="venues.html">Venues</a></li>
   <li><a href="AA/competition.html">Competition</a></li>
   <li><a href="events">Calendar</a></li>
   <li><a href="AA/insurance.html">Insurance</a></li>
   <li><a href="members/index.html">Members</a></li>
 </ul>
 </div>

 <div id="border">
    <div class="Top">
    <div class="TopRight">
    <div class="TopLeft"></div></div></div>

    <div class="Left">
    <div class="Right">

    <div class="contents">

    <div class="LeftPadding1"> </div>
    <div class="LeftPadding2"> </div>
    <div class="LeftPadding3"> </div>
    <div class="LeftPadding4"> </div>
    <div class="LeftPadding5"> </div>
    <div class="LeftPadding6"> </div>
    <div class="LeftPadding7"> </div>
    <div class="LeftPadding8"> </div>
    <div class="LeftPadding9"> </div>

    Toowoomba Company of Archers Inc. (TCA)
    is a not for profit sports club
```

. . .

Figure 3.12 shows the page without style commands and Figure 3.13 shows the page with style commands.

Exercise 3.20: Explain why and how the first dozen lines of the page of Example 3.19 are indented.

Exercise 3.21: Load the page from Example 3.19 into your browser and experiment changing the dimensions of your browser page. How does the formatting change for the page? Change the default size of the browsers fonts—how is the page style affected?

Absolute Positioning Scheme

A box is removed from the normal flow entirely and assigned a position with respect to a "containing block".

position This property specifies how a box is to be positioned relative to the "containing block". It can have the values:

- `static` The box is a normal box, laid out according to the normal flow
- `relative` The box is offset relative to its normal position. When a box is relatively positioned, the position of the following box is calculated as though it were not offset.
- `absolute` The box is offset relative to the box's containing block. Absolutely positioned boxes are taken out of the normal flow.
- `fixed` The box is fixed with respect to the view-port and will not scroll. Fixed boxes are taken out of the normal flow.

Box offsets can be specified using absolute lengths or relative percentages. Percentages are relative to the "containing block".

left This property specifies how far a box's left edge is offset to the right of the left edge of the box's "containing block".

right This property specifies how far a box's right edge is offset to the left of the right edge of the box's "containing block".

top This property specifies how far a box's top edge is offset below the top edge of the box's "containing block".

bottom This property specifies how far a box's bottom edge is offset above the bottom of the box's "containing block".

With absolute positioning the definition of what is the "containing block" changes depending on the absolute positioning method (See Chapter 10 of the CSS2 reference)—

- If the positioning is *relative* or *static* then the "containing block" is the block-level, table-cell or inline-block of the nearest ancestor.
- If the positioning is *fixed* the containing block is the browser window or the page area (for print media).

- If the position is *absolute* the containing block is the nearest ancestor with a position of *absolute*, *relative* or *fixed*.

3.9.1 Classification Properties

These properties classify elements into categories more than they set specific rendering properties.

The properties that follow are not complete. For a complete list of classification properties consult the CSS standard. Only some of the following properties are implemented by browsers. Which property is recognised by which browser is a matter of trial and error.

display This property describes how an element is to be displayed on the page. If the value is `block`, the element is displayed within a new separate box. Examples of block elements are `H1`, and `P`. A value of `list-item` is similar to `block` except that a list item marker is added. The element `LI` is an example of a `list-item`. A value of `inline` results in a new inline box on the same line as the previous content. A value of `none` turns off the display of the element.

A value of `none;` for the `display` property is useful for navigation menus—all pages can have all menus but only the relevant menus are made visible. This is extremely useful as a default template page can be constructed for a site and only the relevant parts are displayed.

white-space This property declares how whitespace inside the element is handled. A value of `normal` means that whitespace is collapsed (the default). A value of `pre` means that whitespace is handled as in the `PRE` element. A value of `nowrap` means text will not be wrapped.

list-style-type This property is used to determine the list-item marker. Possible values are `disc`, `circle`, `square`, `decimal`, `lower-roman`, `upper-roman`, `lower-alpha`, `upper-alpha`, and `none`.

list-style-image This property sets the image that will be used as the list-item marker.

list-style-position The value of this property determines how the list-item marker is drawn with respect to the text in the list item. A value of `inside` means that the text aligns with the marker, a value of `outside` means that the marker is to the left of the aligned text.

Example 3.22: Absolute positioning and modifying the display property of elements is used in the Dynamic Menus example that can be found in the CSS Examples page on the course web site.

3.10 `DIV` and `SPAN` Elements

The HTML element `DIV` is a block level element that does little in itself. Used in conjunction with property styles the `DIV` element becomes a powerful tool for changing the styles of text blocks.

The text level element SPAN, like DIV does little in itself. This element does not have any attributes, but used in conjunction with style properties can change text level styles.

The elements DIV and SPAN where introduced into the HTML language purely as a way off adding style to the rendered document where an existing logical element is not really appropriate.

Example 3.23: An example of using the generic block element DIV to modify the style of its contents—

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
      "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>Rime of the Ancient Mariner</title>
 <style>
<!--
  div.poem { text-align: center;
             line-height: 150%;
             font-size: larger;}

  div.poem pre { font-family: serif;
                 margin-bottom: 2em;}

  div.author {font-style: italic;
              text-align: right;}
->
  </style>
</head>
<body>
<div class="poem">
<h1>The Rime of the Ancient Mariner</h1>
<h3>Part IV</h3>
<pre>
&ldquo;I fear thee, ancient Mariner!
I fear thy skinny hand!
And thou art long, and lank, and brown
As is the ribbed sea-sand.
</pre>
<pre>
I fear thee and thy glittering eye,
And thy skinny hand, so brown.&rdquo;
&ldquo;Fear not, fear not, thou Wedding-Guest!
This body dropped not down.
</pre>
<pre>
Alone, alone, all, all alone,
Alone on a wide wide sea!
And never a saint took pity on
My soul in agony.
</pre>
<pre>
The many men, so beautiful!
And they all dead did lie:
```

The Rime of the Ancient Mariner

Part IV

"I fear thee, ancient Mariner!
I fear thy skinny hand!
And thou art long, and lank, and brown
As is the ribbed sea-sand.

I fear thee and thy glittering eye,
And thy skinny hand, so brown."
"Fear not, fear not, thou Wedding-Guest!
This body dropped not down.

Alone, alone, all, all alone,
Alone on a wide wide sea!
And never a saint took pity on
My soul in agony.

The many men, so beautiful!
And they all dead did lie:
And a thousand thousand slimy things
Lived on; and so did I

...

Samuel Taylor Coleridge

Figure 3.14: One possible rendering of the style of Example 3.23.

```
And a thousand thousand slimy things
Lived on; and so did I
</pre>
<pre>
&hellip;
</pre>
</div>

<div class="author">
Samuel Taylor Coleridge
</div>
</body>
</html>
```

Figure 3.14 shows the rendered page.

Note

CSS gives so much power to the author with user defined classes, that authors could design their own "document language" based on `DIV` and `SPAN`, by assigning style information through the "class" attribute.

Authors should avoid this practice since the elements of HTML have recognised and accepted meanings where `DIV` and `SPAN` with author-defined classes will not.

Do not over-use `DIV` and `SPAN`!

3.11 Questions

Short Answer Questions

Q. 3.24: What is a *property* in the style sheet language. Give an example.

Q. 3.25: Why have tables been used as layout engines in HTML?

Q. 3.26: Why should Style Sheets be used where possible?

Q. 3.27: Why can frames cause navigation problems?

Q. 3.28: Write a basic frameset document.

Q. 3.29: Write a basic 2x2 table.

Q. 3.30: Why are frames given "names"?

Q. 3.31: What is the rendering difference between the `TD` element and the `TH` element?

Q. 3.32: What is a *selector* in the style sheet language? Give an example.

Q. 3.33: How is the `SPAN` element used within HTML documents?

Q. 3.34: How are browser windows "targetted"?

3.12 Further Reading and References

- The examples directory for this module.
- The World Wide Web consortium HTML4.0 definition can be found in the course resources directory. This definition of HTML has been implemented by all browsers.
- The World Wide Web consortium XHTML1.0 definition can be found in the course resources directory. This definition of XHTML has been implemented by all browsers.
- The World Wide Web consortium Cascading Style Sheets (Level 2) definition can be found in the course resources directory.

Department of Mathematics and Computing, USQ
(This file created: June 12, 2012)

Chapter 4 Graphics

The Web for most people is a "visual medium". Correct image use on the Web is not just a matter of design and taste. It requires an understanding of the various image formats, so a designer can use the correct format in the correct situation without compromising quality.

This chapter will discuss the technical details of the major image formats used in the Web and the HTML commands used for displaying images and image-maps.

Chapter contents

4.1 Pixels and Colour

A monitor (computer, TV, projector etc.) is composed of numerous small dots or *pixels* (picture elements). On an LCD screen each pixel is made up of three sub-pixels—a red sub-pixel, a green sub-pixel and a blue sub-pixel (RGB). Ultimately, all image formats must describe the image in terms of `pixels`.

The RGB colour palette is considered an *additive* form of colour, because the red, green, and blue light from the sub-pixels in equal amounts "add" up to white light. By controlling the relative intensity of the three primary colours of the sub-pixels, arbitrary colours can be generated.

Note

The RGB colour cube is used for monitors—that is transmitive colour devices. Reflective colour devices—paper, the most common colour scheme is cyan, magenta, yellow, and black (CMYK) The colours seen on the printed paper are the parts of the spectrum reflected back to your eyes as white light hits the page. The CMYK colour palette is a *subtractive* colour palette since, if you mix the cyan, magenta, and yellow, they should absorb all the colour from white light and you should get black. Due to impurities in all printing inks this does not happen, you don't actually get black, which is why black (or K) ink must be added.

The terms *bit-depth* or *colour-depth* are the terms used to describe the number of bits used to describe colour in an image or on a monitor. For a monitor the limitation on the number of bits used for each pixel are dependent on the monitor and the video card's memory. For an image it is the image file format.

CSS defines colour using a set of predefined colour names or three decimal numbers in the range 0–255 or 3 hexadecimal numbers in the range 00–FF. The range 0–255 means the default colour depth is 24-bit—8-bits for each colour channel. Table 4.1 lists the predefined CSS colour names.

Note

An important extension of RGB colour is RGBA—or 32-bit colour. The extra channel is used as a transparency mask for each pixel. On a monitor RGBA does not make sense as it is not transparent (when we get transparent monitors it will make sense)[1] For image formats RGBA is extremely useful when images are being overlaid.

An understanding of bit depth is important for a web page author—as the bit depth of a visitor's monitor affects colour reproduction, and the manipulation of bit depth in images can be used to significantly decrease file size—and thus decrease download time.

The quality of a displayed image depends upon the quality of the original image file and the display hardware. A major influence is the number of discrete colour intensity levels, and the number of distinct colours, which are used both to encode and to display the image. Our eyes perceive a continuous range of intensity and colour in an image. Digital computers can only reproduce discrete values, but if the system can display a very large number of intensity and colour levels then it is possible to produce a high quality image which compares well with human perception.

Note

Tests have shown that humans can not perceive colour variation beyond 10 bits per channel—that is 30-bit true colour. This implies that 8-bit per channel RGBA colour will be around for a long time.

The number of colours and intensities present in an image file depends upon the image creation hardware (e.g. scanner, digital camera), limitations of the image format, and the way that the image has been processed by image translation tools.

The number of colours and intensities visible in any image on the computer screen depends upon the display hardware (monitor, video card), and operating system configuration.

[1] Video cards that advertise 32-bit colour are not increasing the colour depth—they still only use 24-bit colour—they are increasing the data speed only. The video bus uses 4-byte words by padding pixel information, packing/unpacking of pixel information is avoided and a faster data transmission occurs.

Table 4.1: Predefined colour names in CSS

Colour	Colour Name	Hex. Equivalent
	AQUA	#00FFFF
	NAVY	#000080
	BLACK	#000000
	OLIVE	#808000
	BLUE	#0000FF
	PURPLE	#800080
	FUCHSIA	#FF00FF
	RED	#FF0000
	GRAY	#808080
	SILVER	#C0C0C0
	GREEN	#008000
	TEAL	#008080
	LIME	#00FF00
	WHITE	#FFFFFF
	MAROON	#800000
	YELLOW	#FFFF00
	ORANGE	#FFA500

4.2 Image Formats

An image format describes the colour and position information necessary to create an on screen image. Ultimately the image format must contain the information necessary to set the monitor's pixel colours. There are two basic image format varieties: *raster* or *vector*.

4.2.1 Raster Formats

In its raw form a raster or *bitmap* image is simply a collection of pixels of different colour values—it mimics the way the hardware displays images. The term *raster* relates to the screen display hardware which paints an image one display line or *raster* at a time. Each raster line is a row of pixels. Due to the large number of pixels and colour information in an image, raw bitmaps can be very large. Data compression schemes are used to reduce the image size

Example 4.1: An uncompressed bitmap image of 800×600 pixels with 24-bit colour information would take about 1MB to store (800×600×24/8 bytes).

Given their potential large size, bit-mapped image formats almost always employ some form of compression. There are two forms of compression *lossless* and *lossy*. Lossless image compression means that the compressed image is identical to the uncompressed image. Since all the data of the image must be preserved, the degree of compression, and the corresponding savings, can be minor—it critically depends on the image itself. Lossy compression, on the other hand, does not preserve the image exactly, and therefore can provide a much higher degree of compression. With lossy compression, since data is discarded images are smaller but the image quality will be degraded.

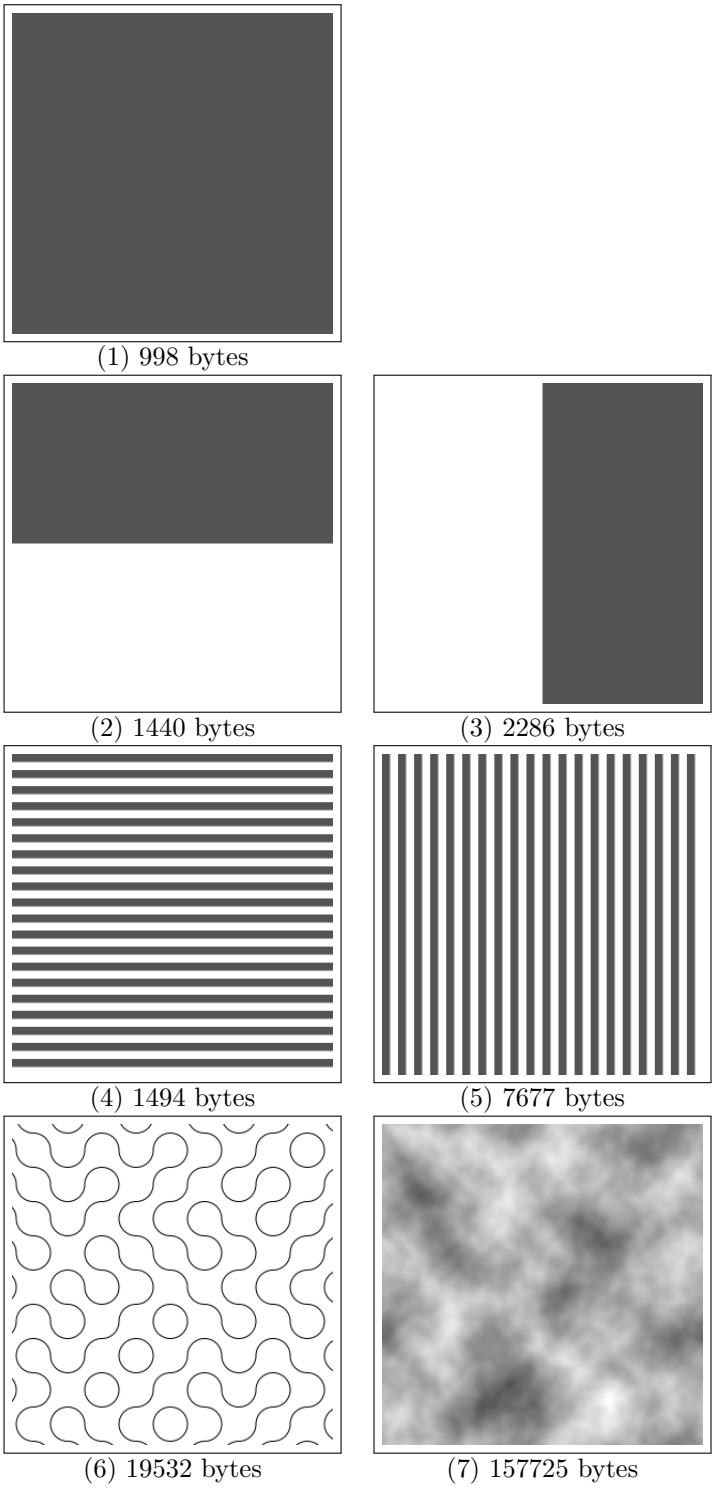
GIF

The Graphics Interchange Format (GIF) allows at most 256 colours per image, where each colour is described by a set of 8 bit red/green/blue intensity values. GIF files keep a colour table (with a maximum of 256 entries) or palette which describes every colour in the image; each pixel in the image is denoted by its table item number (an 8 bit value) rather than its 24 bit colour intensity value. Sometimes this is referred to as *indexed colour*. This leads to a compact representation at the expense of a reduced set of possible displayable colours.

Compression GIF images are compressed—using the Lempel-Ziv-Welch (LZW) lossless compression scheme. The use of a colour table together with compression results in a good compromise between image quality and storage size. There are two GIF standards: GIF87a and the later GIF89a. GIF89a added support for transparency and animation.

The basic form of compression employed is called *run-length encoding* (RLE). RLE looks for pixels with the same value and replaces the "run" of identical pixels with one pixel and a count. This lossless

Figure 4.1: Demonstration of the GIF compression algorithm for a 600×600 pixel image



(1) 998 bytes

(2) 1440 bytes

(3) 2286 bytes

(4) 1494 bytes

(5) 7677 bytes

(6) 19532 bytes

(7) 157725 bytes

Figure 4.2: Example of aliased and anti-aliased text

compression works well with large areas of continuous colour. Figure 4.1 shows the GIF compression scheme in practice. As can be seen the run direction searched is along the horizontal direction (raster by raster). Rotating an image can increase its size significantly.

Given the GIF formats difficulty in dealing with variability in images, the format should only be used for illustrations and any images that contain large amounts of continuous colour.

Dithering Since the GIF format can only represent 256 colours simultaneously—any 24-bit colour image converted to GIF must reduce the number of colours in the image—millions of colours must be remapped to 256. When remapping from a large number of colours to a smaller colour palette *dithering* occurs. The process of dithering attempts to create a colour outside the palette, by a diffusion of coloured pixels from within the palette. Thus, creating the illusion of the missing colour. Depending on the algorithms used to quantise a 24-bit image to a 256 colour palette the new smaller GIF image can look very poor indeed.

Transparency GIF images also support the concept of *transparency*. One bit of transparency is allowed, which means that one colour can be flagged as transparent. Transparency allows the background that an image is placed upon to show through. The problem with GIF transparency is that it is 1-bit transparency—either on or off—which causes problems with *anti-aliasing*. Anti-aliasing attempts to smooth pixelated or jagged edges by blending one colour into another. Figure 4.2 shows the smoothing effect anti-aliasing has on text. If the transparent colour has been blended on colour edges then a halo effect will be created when transparency is used. Figure 4.3 demonstrates this halo effect.

Exercise 4.2: Discuss two ways to avoid the anti-aliasing 1-bit transparency halo problem using the GIF format.

Interlaced It can be frustrating to wait for an entire image to be loaded before seeing it in completion. To alleviate this problem, the

Figure 4.3: The halo effect created by anti-aliasing and 1-bit transparency

GIF format allows *interlaced* images. These are standard GIF images but are not stored in a sequential-order from top to bottom. The actual algorithm is

(a) starting with row 0, store every 8th row

(b) starting with row 4, store every 8th row

(c) starting with row 2, store every 4th row

(d) starting with row 1, store every 2nd row

The advantage of interlaced images is that the user can quickly get an idea of what the image looks like without waiting for the entire image to download—thus avoiding user frustration.

Animation The GIF format also supports animation. This works by stacking GIF image after GIF image into the file which are then displayed in sequence—in the manner of a flip-book. All the image frames of the animation share the same colour index table—so there is a maximum of 256 colours for the entire animation. The GIF animation extension also allows timing and looping information to be added to the image. Most graphics programs support combining multiple images into an animation.

Since the GIF animation is basically image after image, the file size is the combined size of all the images in the animation—which can produce a surprisingly large file.

PNG

PNG (Portable Network Graphics—pronounced 'ping') is a more recent alternative to the GIF format. It was devised as a second generation better alternative to the GIF format[2]. PNG can provide a GIF-like indexed format, a 24-bit true colour format, or a 32-bit RGBA format (true colour with an 8-bit transparency channel). Compression is performed without loss of information and provides better rates of compression than the GIF format. The PNG compression is a two stage process. First each image line is filtered so that hopefully the transformed image is more easily compressed. The non-patented

[2] Development on an alternative to GIF was started in 1994 when Unisys Corp. enforced its copyright on the Lempel-Ziv-Welch compression algorithm used in GIF images.

Table 4.2: Comparing the size of PNG and GIF files using the images of Figure 4.1

Image	GIF	PNG
(1)	998	229
(2)	1440	256
(3)	2286	350
(4)	1494	355
(5)	7677	362
(6)	19532	8445
(7)	157725	99768

Table 4.3: The five filtering algorithms used to predict the current byte

Filter	Predicted Byte
None	Raw byte value passes through unaltered
Sub	Byte to the left (A)
Up	Byte above (B)
Average	Average of bytes A and B (rounded down)
Paeth	A, B or C (above-left byte) whichever is closest to A+B-C

lossless compression algorithm "deflate" is then used to compress the transformed image.

Generally PNG files are smaller than GIF files. Table 4.2 compares the PNG and GIF file sizes of the images from Figure 4.1.

Pre-Filtering The pre-filtering employed by PNG scans the image line by line and tries to predict the value of each byte by using previously scanned neighbouring bytes. The predicted byte value is then subtracted from the actual byte value. If the prediction is good then the final filtered byte value should be zero[3]. An image line filtered in this way is often more compressible than the raw image line would be, especially if it is similar to the line above. Filtering is done on byte values not pixels—as pixels could be represented by 1, 2, 3, 4 or more bytes. Table 4.3 lists the five filter algorithms used to predicted the current byte.

Interlaced PNG offers an additional interlacing scheme as well as the GIF 4-pass scheme. It is a 2-dimensional, 7-pass scheme. The two dimensional scheme allows a clearer low-resolution view of the image to be visible earlier in the image download—unfortunately it also tends to reduce the image's compressibility.

Animation PNG supports animation through its Multiple-image Network Graphics (MNG) extension—unfortunately it is not well supported by browsers—though plugins are available.

[3] This filtering method is not surprisingly called 'Method 0'.

JPEG

The Joint Photographic Experts Group (JPEG)[4] format, unlike GIF and PNG is a *lossy* format. It was designed to compress photographic images that may contain millions of colours or shades of grey. Since the JPEG format is lossy, there is a trade-off between image quality and file size. The underlying format for JPEG is 24-bit colour but because of the lossy compression the format can store images in files significantly smaller than the GIF or PNG format.

Compression The JPEG compression algorithm is lossy—but it is designed so that the human eye will (hopefully) not notice the missing information. The algorithm reduces spatial colour variation but not the brightness of the image—the human eye can see significantly more fine detail in the brightness of an image than the colour of an image. The modified image is then split into 8×8 pixel block—each channel in the block is converted into a frequency representation using a discrete cosine transform (that is the data is now represented by a sum of cosines with different frequencies). The human eye is good at seeing small differences in brightness over a relatively large area, but not so good at distinguishing the exact strength of a high frequency brightness variation. This means the coefficients of cosines with high frequencies can be reduced hopefully to to zero. This reduces the amount of information that has to be stored for each block and thus stored for the entire image.

Reduce the quality of the image—by setting more cosine coefficients to zero—which also reduces the size of the file.

Transparency JPEG images do not support transparency.

Progressive JPEG images support a feature called *progressive* JPEG. Progressive JPEGs behave similarly to interlaced GIF or PNG images. Progressive JPEGs when displaying progress from low resolution to their final high resolution, going from a fuzzy, blurred image to the final clear image.

Animation JPEG images do not support animation.

Note

Choosing between PNG, GIF or JPEG is usually straightforward: if you wish to display a photograph use JPEG, if it is an illustration use PNG (or GIF).

4.2.2 Vector Formats

An alternative to *raster* formats are *vector* or *geometric* formats. A vector format describes the image using mathematical curves, shapes, points and colours. It can be very compact as only the curves and regular shapes need be described not each pixel. On the other hand, a *rendering* algorithm or engine must be used to convert from the

[4] The name of the committee that wrote the standard

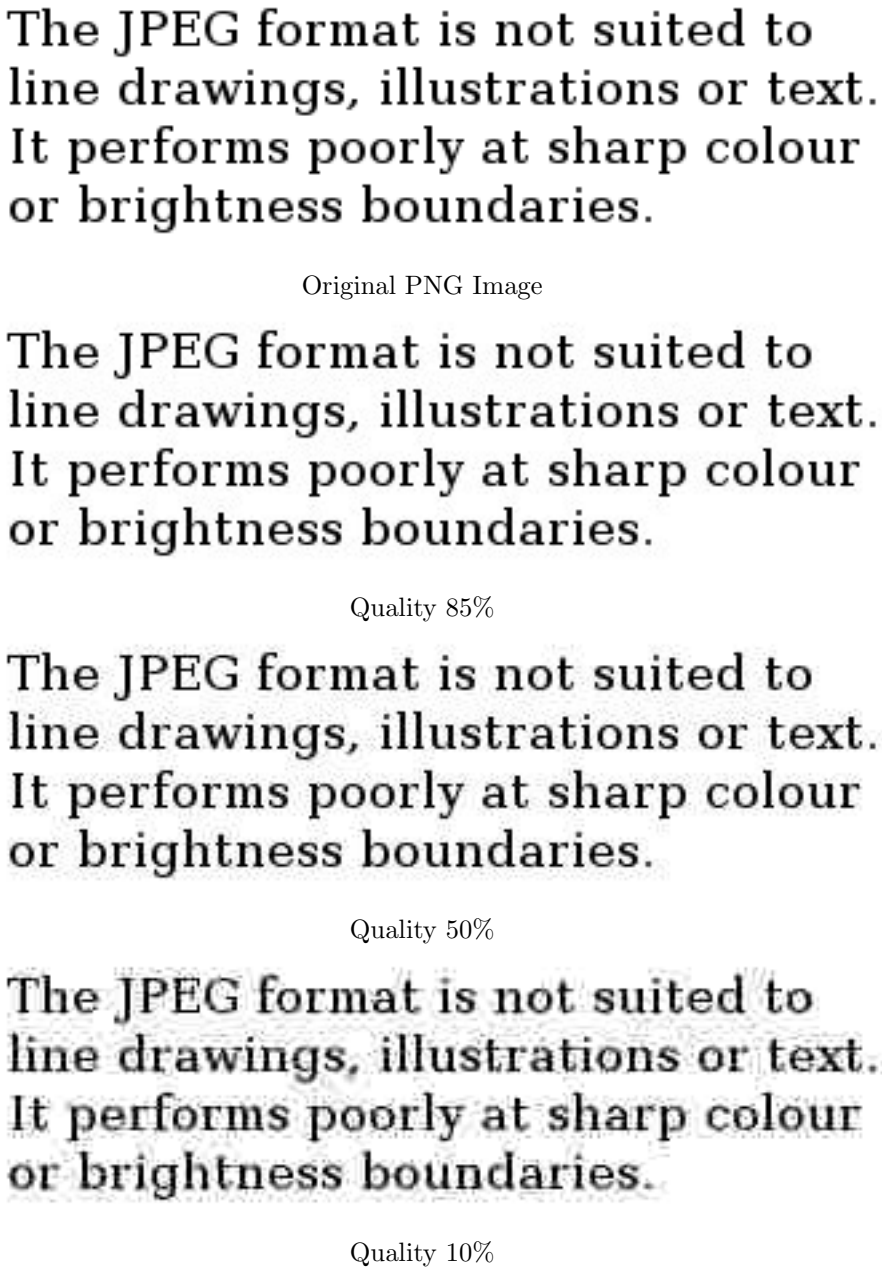


Figure 4.4: Three JPEG images showing the discrete cosine artifacts growing as the quality is decreased. Even with the quality at 85% shadows around the letters are visible.

geometric description of the image to the raster image before it can be displayed—this can take time.

One major advantage of vector images is that they can be scaled easily preserving their smooth shape—unlike raster images.

Most vector formats (SVG, Postscript, PDF etc.) also support the inclusion of raster images within the vector image.

Note

Vector formats are the preferred format for text since they are scalable. Each character in a font description file is described by a set of Bézier curves—which are easily added to a vector image—and preserve the character shapes at any size.

It should be noted however, that high quality characters are designed specifically for a very narrow size range and should not be scaled to any size. The reason is that the eye is not a linear instrument what is visually pleasing and clear at 10 points scaled to 20 points will look odd—typically the white space around and within characters is too great.

Adobe Corporation's professional fonts normally come in four recommended size ranges (the ranges are different for each design)—

- footnote, superscript, subscript size ($\sim 6-9$ points)
- text size ($\sim 10-14$ points)
- heading size ($\sim 15-20$ points)
- display size ($\sim >20$ points)

Using a font scaled outside its recommended size range produces very strange looking text.

Postscript

Unlike other vector image formats Postscript is a "programming" language—not a mark-up language. PostScript is an interpreted, stack-based language where the language syntax is "reverse-Polish" notation. Postscript has been used extensively as an "interpreter" within Laser Printers (which are raster devices). By placing an interpreter within the printer has a number of advantages—

- Offload the rasterization of the vector description to the printer's CPU.
- Possibility of using a single language that could be available on any printer—thus precluding the need for individual drivers for each printer.
- Since Postscript is a complete printing language it can be displayed on any device with an interpreter—making it a device independent language.

Example 4.3: Postscript uses a "Reverse Polish" notation (also called postfix notation). In Reverse Polish notation the operators (procedures) follow their operands (parameters)—

```
1 0 0 setrgbcolor
```

Set the current draw colour to Red. The procedure `setrgbcolor` requires three numbers to set the RGB colour space. The procedure pops 3 numbers of the top of the stack and uses those numbers for the new colour.

Example 4.4: The following is part of a small Postscript file. (The full version can be found on the course web site.)

Figure 4.5 shows the rendered image of the full file.

```
%!PS-Adobe-3.0 EPSF-3.0
%%Creator: inkscape 0.46
%%Pages: 1
%%Orientation: Portrait
%%BoundingBox: 130 121 415 313
%%EndComments
%%Page: 1 1
0 480 translate
0.8 -0.8 scale
0 0 0 setrgbcolor
[] 0 setdash
1 setlinewidth
0 setlinejoin
0 setlinecap
gsave [1 0 0 1 0 0] concat
gsave
% Create a rectangle and fill it with Blue
0 0 1 setrgbcolor
newpath
229.06523 211.12801 moveto
477.63666 211.12801 lineto
477.63666 412.55658 lineto
229.06523 412.55658 lineto
229.06523 211.12801 lineto
closepath
eofill
grestore
% Draw a Red Border arround the Blue rectangle
1 0 0 setrgbcolor
[] 0 setdash
3 setlinewidth
0 setlinejoin
0 setlinecap
newpath
229.06523 211.12801 moveto
477.63666 211.12801 lineto
477.63666 412.55658 lineto
229.06523 412.55658 lineto
229.06523 211.12801 lineto
closepath
stroke
gsave [1 0 0 -1 162.59674 324.0643] concat
gsave
% Print Text.
/newlatin1font {findfont dup length dict copy dup
                /Encoding ISOLatin1Encoding
                put definefont} def
/BitstreamVeraSans-Roman-ISOLatin1
/BitstreamVeraSans-Roman newlatin1font
24 scalefont
```

```
setfont
0 0 0 setrgbcolor
newpath
0 0 moveto
(         ) show
grestore
grestore
gsave [1 0 0 -1 162.59674 324.0643] concat
gsave
/BitstreamVeraSans-Roman-ISOLatin1 findfont
24 scalefont
setfont
0 0 0 setrgbcolor
newpath
0 0 moveto
(An Example of SVG Graphics) show
grestore
grestore
%% ... lines removed from original file
%% ... full file available on course web site.
showpage
%%EOF
```

Portable Document Format (PDF)

The PDF is a file-format and markup-language that describes a fixed-layout document that includes text objects, font objects, image objects, and vector graphic objects.

To layout the page and render vector graphics PDF uses a subset of the Postscript language—the flow control aspects of Postscript have been removed and only the simple graphics commands such as move, draw and fill are retained.

Example 4.5: The following is part of a small PDF file. (The full version can be found on the course web site.) Figure 4.5 shows the rendered image.

This example shows how PDF encodes objects as binary data streams.

```
%PDF-1.4
3 0 obj
<< /Length 4 0 R
   /Filter /FlateDecode
   /Type /XObject
   /Subtype /Form
   /BBox [ 0 0 480 480 ]
   /Group <<
      /Type /Group
      /S /Transparency
      /CS /DeviceRGB
   >>
   /Resources 2 0 R
>>
stream
x^K^Em\305\375T=O$1...
endstream
endobj
...
```

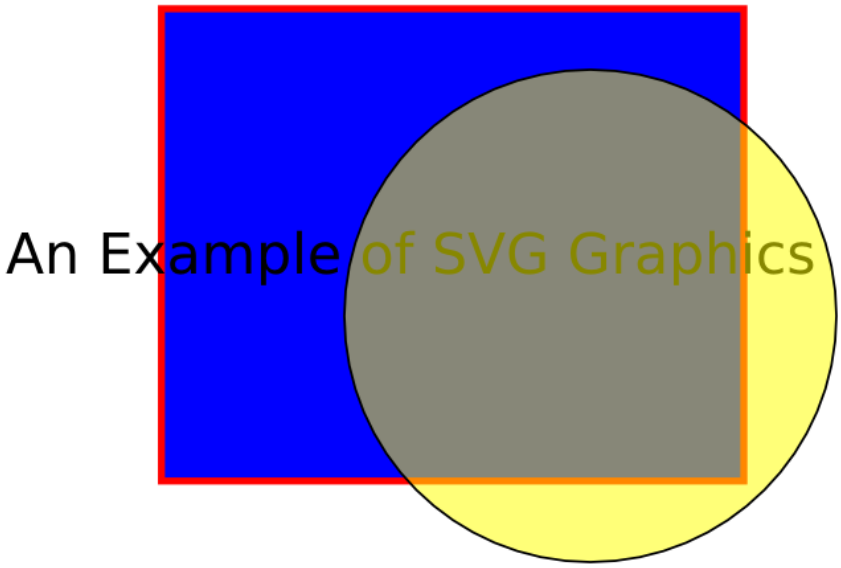


Figure 4.5: The rendered image produced by the vector code of Examples 4.4,4.5, or 4.6,

SVG

Scalable Vector Graphics (SVG) is an XML markup-language developed by the World Wide Web Consortium (W3C) to describe two-dimensional vector images. Most Web browsers support SVG images natively. SVG recognises three types of graphics object—vector, raster and text.

SVG recognises basic shapes such as Lines, Rectangles, Circles, Ellipses, and Paths. All shapes are defined by a set of (x, y) coordinates.

Example 4.6: The following is a small SVG file. Figure 4.5 shows the rendered image.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- Created with Inkscape (http://www.inkscape.org/) -->
<svg xmlns:svg="http://www.w3.org/2000/svg"
     xmlns="http://www.w3.org/2000/svg"
     version="1.0"
     width="744.09448"
     height="1052.3622"
     id="svg2">
  <defs id="defs4" />
  <rect width="248.57143" height="201.42857"
        x="229.06523" y="211.12801"
        id="rect2385"
        style="fill:#0000ff;fill-opacity:1;
               fill-rule:evenodd;
               stroke:#ff0000;stroke-width:3;
               stroke-linecap:butt;stroke-linejoin:miter;
               stroke-miterlimit:4;stroke-dasharray:none;
               stroke-opacity:1" />
  <text x="162.59674" y="324.0643"
        id="text3163"
```

```
        xml:space="preserve"
        style="font-size:24px;font-style:normal;
               font-variant:normal;
               font-weight:normal;font-stretch:normal;
               text-align:start;line-height:125%;
               writing-mode:lr-tb;text-anchor:start;
               fill:#000000;fill-opacity:1;stroke:none;
               stroke-width:1px;stroke-linecap:butt;
               stroke-linejoin:miter;stroke-opacity:1;
               font-family:Bitstream Vera Sans;">
        <tspan x="162.59674" y="324.0643"
        id="tspan3167">An Example of SVG Graphics</tspan>
    </text>
    <path d="M 541.98912,400.8461
          A 105.01039,105.01039 0 1 1 331.96834,400.8461
          A 105.01039,105.01039 0 1 1 541.98912,400.8461 z"
          transform="translate(-24.841167,-58.715487)"
          id="path2385"
          style="fill:#ffff00;fill-opacity:0.5288889;
                 fill-rule:evenodd;
                 stroke:#000000;stroke-width:1px;
                 stroke-linecap:butt;stroke-linejoin:miter;
                 stroke-opacity:1" />
</svg>
```

Exercise 4.7: Explain why converting a PNG image into an SVG image will not mean that the original image is now infintely scalable?

4.3 Images as Anchors

There are a number of ways in which images can be employed as anchors for hypertext link. The first and simplest is to wrap an image element in an HTML anchor. For instance, the HTML code below results in an image being displayed; clicking anywhere in the image will activate the link.

```
<a href="http://jigsaw.w3.org/css-validator/check/referer">
   <img width="88px" height="31px" title="Valid CSS"
        src="valid-css2-blue.gif" alt="Valid CSS!"
        style="border: 0px" /></a>
```

Notice that this is simply a combination of the standard anchor `<A>` and image `<IMG>` tags. This technique can be used, for instance, to create navigation buttons at the bottom and top of a page. An array of buttons or individual images could be arranged using the HTML table element or CSS commands.

Note

The `style="border: 0px"` style command removes the small border which normally surrounds anchored images when the browser is configured to underline links.

A more sophisticated and general method of utilising an image as an anchor involves creating an image map. An image map associates a set of (image region, URL) pairings with a particular image. The region (sometimes called a *hot spot*) defines a part of the image; clicking on that part activates the associated link. There are two ways of programming image maps: *server-side maps* and *client-side* maps.

Both schemes are logically equivalent, but are described differently, and have different resource usage implications.

With a server-side map (historically the first kind of map implemented), the user's mouse-click is seen by the browser and the coordinates of the mouse pointer are sent to the server. The server looks up the image map to determine the URL, which it transmits back to the user's browser so that the link can be activated. Obviously this scheme imposes a network communication overhead (two messages sent between client and server), as well as computational load on the server, whose job it is to look up the URL given image coordinates.

A neater solution uses client-side maps. Here the page containing the image also contains an image map specification. So now the client's browser has all the information necessary to calculate the URL, thus eliminating the load on both the network and the remote server.

Client-side maps have largely superseded server-side maps as they are easier to set up and are more economical of server and network resources. They also have the distinct advantage of providing feedback to the user via the information bar in their browser and also the pointer status when the mouse pointer is positioned over a hot spot.

4.3.1 Server-side Image Maps

Here are the steps required to create a server-side image map.

(a) Set up the server to allow server-side image maps.

(b) Select an image. This is fairly straightforward. Make sure that the image is not too big (we don't want the user to have to scroll their window). Also make sure that it displays clearly (i.e. it is not too small to use).

(c) Create an image map.

(d) Write the HTML markup command to include the image.

Setting up the server

The server must be configured correctly before it will interpret image maps. Your server is already configured to interpret image maps. Configuration of the server will be covered in a later section[5].

Creating an image map

A separate map file must be created. The file contains a set of entries; there is one entry (one line) for each active region. The general format is

[5] For those interested now, here is what to do for the Apache server. The idea is that a *handler* (a piece of code within the Apache server) must be invoked to deal with the map file. Add this line to your `httpd.conf` file

```
AddHandler imap-file map
```

This says to use the handler module "`imap-file`" when a map file is encountered. You should also be able to add the directive to a directory access file in your document directory. See the later section on server configuration for more information.

region-type URL coordinates

Regions can be rectangles, points, circles, irregular polygons, or default.

The URL can describe any local or non-local item. It can be another HTML document, an image, or anything which can be legally described by a URL.

The region keywords and coordinates are summarised in table 4.4. X,Y coordinates are expressed as pixel measurements from the upper left corner of the image.

Table 4.4: Image map directives

Directive	Coordinates	Example
`rect`	Two space separated coordinate pairs defining top left and bottom right corners.	`rect URL 20,30 45,100`
`circle`	Two space separated coordinate pairs defining centre and any edge point.	`circle URL 20,30 45,100`
`poly`	Series of (max 100) space separated coordinate pairs defining set of vertices of a closed polygon. If first and last point are not the same, then the map will assume that they are connected.	`poly URL 20,30 45,100 55,50 30,5`
`point`	A single coordinate pair. If a mouse click does not fall within any other region, then the closest point is selected.	`point URL 20,30`
`default`	No region is specified. The default URL is returned when no region is selected and there are no points present in the map.	`default URL`

HTML markup

Now the image can be added to a HTML document. As an example, the following HTML

```
<a href="light-index.map">
  <img src="light-index.jpg" alt="index of lighthouses" ismap />
</a>
```

Where `light-index.map` is

```
rect aug3.jpg 9,3 51,85
rect canaveral1.jpg 65,3 111,84
rect griffiths.jpg 120,7 171,86
rect lhse2.jpg 176,5 230,84
rect lhse3.jpg 234,9 287,85
rect lhse5.jpg 293,8 346,85
rect lhse60.jpg 4,96 56,173
rect lhse7.jpg 62,96 112,172
rect otway2.jpg 119,96 171,172
rect pcola1.jpg 175,96 230,173
```

```
rect ponce1.jpg 237,93 284,172
rect ponce3.jpg 294,105 345,170
```

displays a thumbnail index to a set of images. See the accompanying web page on the course web site.

Note

This scheme is very similar to that of the simple anchored image described earlier. The only difference is that the `ismap` directive says to interpret the hyper-link `light-index.map` as a map.

Relative URLs

Relative URLs appearing in the map file are normally interpreted *relative to the location of the map file*. Thus in the example above, the images are expected to be in the same directory as the map file.

The Apache server allows you to specify how relative URLs in a map file are to be interpreted. The optional `base_uri` directive in the map file does this.

`base_uri map`	URLs are relative to the map file
`base_uri referer`	URLs are relative to the html file in which the image reference appears
`base_uri` *`specific-URL`*	URLs are relative to the specified (local or non-local) URL

Menus and further information

Using the Apache server, it is possible to arrange for a menu to appear if no region is selected in an image. See the description of the module `mod_imap` in the Apache documentation. It also contains more extensive details about map file formats.

4.3.2 Client-side image maps

Client-side image maps are almost identical in concept to server-side maps. You must create a map, but for Client-side image maps this map is included in the HTML file which refers to the image, not in a separate file.

Using the same example image as for server-side mapping results in the following client-side image map.

```
<img src="light-index.jpg" alt="index of lighthouses"
     usemap="#index" />

<map name="index">
<area shape="rect" coords="9,3,51,85" href="aug3.jpg"/>
<area shape="rect" coords="65,3,111,84" href="canaveral1.jpg" />
<area shape="rect" coords="120,7,171,86" href="griffiths.jpg" />
<area shape="rect" coords="176,5,230,84" href="lhse2.jpg" />
<area shape="rect" coords="234,9,287,85" href="lhse3.jpg" />
<area shape="rect" coords="293,8,346,85" href="lhse5.jpg" />
<area shape="rect" coords="4,96,56,173" href="lhse60.jpg" />
<area shape="rect" coords="62,96,112,172" href="lhse7.jpg" />
<area shape="rect" coords="119,96,171,172" href="otway2.jpg" />
<area shape="rect" coords="175,96,230,173" href="pcola1.jpg" />
```

```
<area shape="rect" coords="237,93,284,172" href="ponce1.jpg" />
<area shape="rect" coords="294,105,345,170" href="ponce3.jpg" />
<area shape="default" nohref>
</map>
```

Note the use of the new HTML tags `map` and `area` to specify a map, and also the new `img` attribute `usemap` which specifies that client-side mapping is being used. The `usemap` attribute specifies the URL of the associated `map` tag. Usually it is in the same document so that it has a form like `#label`, but it can be in any file as long as it is accessible.

Four different shapes are possible. These have the same role as those of the server-side map, but the syntax is a little different. The main difference, apart from the fact that the information is encoded as area attributes, is that lists of coordinates are separated by commas.

Typical examples are

```
<area shape="rect" coords="20,30,45,100" ...  />
<area shape="circle" coords="20,30,15" ...  />
```

Note that the numbers are X,Y,Radius (unlike server side)

```
<area shape="poly" coords="20,30,45,100,55,50,30,5" ...  />
<area shape="default" nohref />
<area shape="default" href=...  />
```

The `nohref` attribute indicates that no action is to be taken.

The `alt` attribute can be used with the `area` tag to give a name to the region. This can be displayed in a status bar or used by non-graphics browsers and applications. For example we could (and should!) have coded the map entries like this:

```
<area shape="rect" coords="65,3,111,84" alt="Cape Canaveral"
      href="canaveral1.jpg" />
```

4.3.3 Creating map descriptions

You can code image maps by hand, using for instance the `display` utility to determine coordinates together with a text editor to actually type in the commands. If using `display`, moving the mouse with the middle button of a three button mouse (right button on two-button mouse) depressed should show you the coordinates.

There is an easier way. There are many programs available to make the creation of image maps easier. One extremely powerful program available under Linux is `The Gimp`. This is an image manipulation program as powerful as Adobe Photoshop.

Start `The Gimp` from the command line using the command `gimp`. Load the image into `The Gimp`, using the menu item `file->open`.

When the image is loaded into gimp right click on the image. Select the menu item `Filters->Web->ImageMap`. The `ImageMap` script can produce client side or server side imagemaps, and graphically allows you to specify the hot regions of the image.

4.4 Questions

Short Answer Questions

Q. 4.8: What is an *image map* ?

Q. 4.9: What is a pixel?

Q. 4.10: Define the term *raster* . In what contexts is it used?

Q. 4.11: What is the main disadvantage of the raster format?

Q. 4.12: What is the difference between *vector* and *raster* formats?

Q. 4.13: What does *rendering* mean? When is it used?

Q. 4.14: Which format *vector* or *raster* is most closely linked to the screen display hardware?

Q. 4.15: Why are Monochrome images more compact than Colour or Greyscale images?

Q. 4.16: Explain the LUT (Look Up Table) of 8 bit colour formats.

Q. 4.17: Explain the conundrum of how the GIF format can define 24 bit colour but can only display 8 bit.

Q. 4.18: What is *lossy* compression?

Q. 4.19: Why is compression necessary in image formats?

Q. 4.20: What is the *most* significant difference between the GIF or PNG and JPEG formats?

Q. 4.21: What is "pre-filtering" used for in the PNG format?

Q. 4.22: Suggest how a GIF image would be increased in size by a factor of 2 in each linear dimension?

Q. 4.23: Why is converting a PNG image to an SVG image a waste of time?

Q. 4.24: Why are Client-side image maps prefered over Server-side image maps?

4.5 Further Reading and References

- `www.w3.org/standards/webdesign/graphics` — the W3C site on graphics standards.
- `www.w3.org/Graphics/` — the W3C site "Graphics on the Web".
- `www.jpeg.org/` — the Joint Photographic Experts Group Home Page.
- `www.libpng.org/pub/png/` — the PNG Home page
- `www.w3.org/Graphics/SVG/` — the W3C SVG site

Chapter 5 Web Design

Web sites are often developed from one particular point of view—it may be content centred, or technology centred, or graphically centred. Web sites are rarely developed from the most important view of all—the user's! Keeping a site's users in mind and always trying to meet their needs should be the primary goal for anyone building a Web site.

Sites should be built for common user capabilities. They should be accessible to all and be able to account for the differences of individuals. This at first glance appears to be contradictory or impossible—it is not, but it is not easy. A site built for users requires thought, iterative design, testing and user feedback. It also requires the designer to be aware of W3C Web standards and guides and most importantly of all the ability to be flexible with cherished design ideas.

Chapter contents

5.1 What is Web design?

There are approximately five facets of Web design that need to be considered when building a new site:

Purpose The reason the site exists! This is arguably the most important consideration when designing a Web site. The purpose of the site should be considered in all decisions when creating it.

Content The information on the site, the way text is written, how it is organised, presented, and structured within a page and across pages.

Visual The page layout used in the site. The graphical elements used either as information, decoration or for navigation.

Technology The use of XHTML, CSS, XML technologies, Javascript, PHP, AJAX, databases etc.

Delivery The speed and reliability of delivering a site's content over the Internet.

The amount each of the five elements influence a site's design will vary according to the type of site being built. A personal home page has very different content than a shopping site. An internal web site for a company will have different visual imperatives than the company's public web site. Precisely what is meant by "Web Design" is very fluid but basic "good" design practices are similar across the entire Web.

5.2 User-Centred Design

A common mistake made in Web development is that, far too often, sites are built more for developers and their needs rather than for a site's actual users. When designing a site the designer must always remember they are not the user! The site developer has an intimate knowledge of the site they have built. They understand where information is. They understand what specialist software the Web browser may require. They have designed it for their hardware characteristics. If a site is built around the developer's skill levels and hardware most users of the site will be very confused. Web site developers must recognise that most users of their sites will not have an intimate knowledge of the site nor have the same expectations for the site as the developers do. The key to successful, *usable* Web site design is always trying to think from the point of view of a "typical" user. Design that puts the user first is called *User-centred design*.

Understanding the users' needs is not easy. Sites should be built for common user capabilities, rather than for the extreme novice or knowledgeable user. Sites should be accessible to all and be able to account for the differences exhibited by individuals or their hardware. While a site should always be built for users, the desires of the site's owners and creators must also be met. The fine balance of power between the users, the creators and the owners is not always easily achieved.

5.2.1 Usability

What exactly does it mean for something to be "usable"? Consider the following definition from ISO 9241 *Ergonomics of Human System Interaction*,

> "[Usability is] the extent to which a product [or Web site] can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use."

Consider this definition in detail—

- Firstly, when talking about usability the user group has to be well defined. Usability will vary greatly depending on the user.
- Secondly, usability should be related to a task. Sites should not be considered *usable* in some general sense—but within the context of a specific task.

- Usability is then judged by the effectiveness, efficiency, and satisfaction the user experiences trying to achieve a specific task—

  Effectiveness: whether or not users are able to achieve their goals. If users are unable to, or only partially able to complete the task they set out to perform at a site—then the site is not usable!

  Efficiency: if users make a great many mistakes or have to perform tasks in a convoluted way when they visit a site—then the site is not usable!

  Satisfaction: The user must be satisfied with the performance and outcomes of the task.

Exercise 5.1: Using the definition of "Usability" above how "usable" are the following sites?

Starting with the task to perform—identify the "User Group" and then rate the Effectiveness, Efficiency and Satisfaction of the site when performing that task.

- Site: http://www.usq.edu.au/
  Task: Enrolling as a new student.
- Site: http://www.usq.edu.au/
  Task: Printing an assignment cover sheet.
- Site: http://www.birch.com.au/
  Task: Booking a movie ticket online.

Many other definitions of usability exist—Jacob Nielsen (See Section 5.6 below), suggests that the following five ideas determine the usability of a site:

- Learnability
- Rememberability
- Efficiency of use
- Reliability in use
- User satisfaction

By this definition, a site is usable if it is easy to learn, easy to remember how to use, reliable in that it works correctly and helps users to perform tasks correctly, and results in the user being generally satisfied using the site.

For Web design both definitions do not appear useful—because people are different and have different levels of capabilities and Web knowledge, not everyone is going to agree on what is useful. A site that is easy for one user may be difficult to understand for another!

There is no absolute description of what constitutes a usable site—the best that can be hoped for are guidelines.

5.2.2 Common User Characteristics

There are no generic people, but people do tend to have similar physical characteristics. Most people tend to see about the same, are capable of remembering things, and react to stimuli in about the same way. But remember people are individuals and what is considered 'similar' covers a very broad spectrum. However, as with all aspects of Web design, you should aim first for the common user and make sure to account for differences.

Vision

The primary way most users access Web data is visually. They look at a screen and process data in the form of text, colour, graphics or video. The user's ability to see is obviously important. Unfortunately, many sites assume that users have super-human vision, because they use very small text, or have little contrast between foreground and background elements, or have text overlaying background images.

Site designers should always keep in mind that they have no way to know the visual acuity of visitors to their site. Text should never be specified in absolute units of pixels or printer's points. When using pixels the designer does not know the physical size of the pixels on the user's monitor and cannot know the final size of the text. When using points the designer does not know how good the users eye sight is. If fonts sizes must be changed (outside the normal changes required for headings) use relative size changes—relative to the user's choice of the base font size.

Example 5.2: Study the "Text Size Example" that can be found in the examples directory of the course web site. This example compares font sizes using pixels, points and logical size definitions.

In order to avoid troublesome colour combinations, site designers should be aware of colour perception. Three factors affect how colour is perceived:

- **Hue** the degree to which a colour is related to the basic colours—red, green and blue.
- **Saturation** the degree to which a colour differs from achromatic (white, grey, or black).
- **Lightness** the degree to which a colour appears lighter or darker than another under the same viewing conditions.

Example 5.3: The Web Content Accessibility Guidelines (WCAG 2.0) of the W3C suggests that the visual presentation of text have a *contrast ratio* between the foreground and background colours of at least 4.5:1.

The *contrast ratio* is defined as—

$$\frac{L1 + 0.05}{L2 + 0.05}$$

where

Figure 5.1: Text with different contrast ratios relative to the background colour.

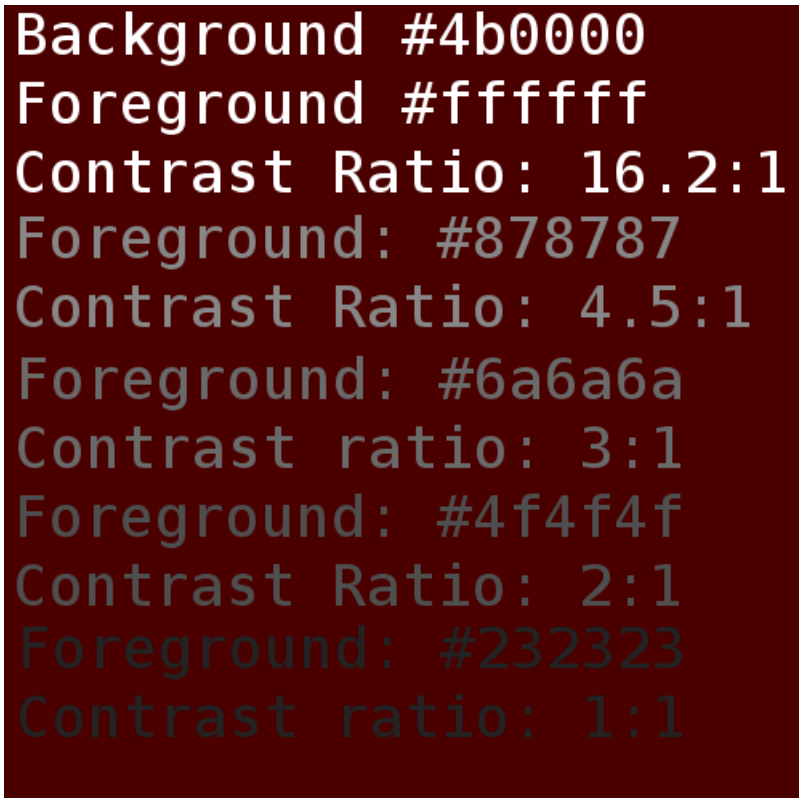


Figure 5.2: Comparison of foreground and background colours of similar hue, saturation and lightness. (note the lightness formula used was $L = (\min(\text{R,G,B}) + \max(\text{R,G,B}))/2$)

- $L1$ is the *relative luminance* of the lighter of the colours, and
- $L2$ is the *relative luminance* of the darker of the colours.

Contrast ratios can range from 1 to 21

The *relative luminance* is the relative brightness of any point in a colour space, normalised to 0 for darkest black and 1 for lightest white. For the sRGB colour space the relative luminance is defined as

$$L = 0.2126\mathcal{R} + 0.7152\mathcal{G} + 0.0722\mathcal{B}$$

where

$$\text{if R} <= 0.03928 \text{ then } \mathcal{R} = \frac{\text{R}}{12.92} \text{ else } \mathcal{R} = \left(\frac{\text{R} + 0.055}{1.055}\right)^{2.4}$$

$$\text{if G} <= 0.03928 \text{ then } \mathcal{G} = \frac{\text{G}}{12.92} \text{ else } \mathcal{G} = \left(\frac{\text{G} + 0.055}{1.055}\right)^{2.4}$$

$$\text{if B} <= 0.03928 \text{ then } \mathcal{B} = \frac{\text{B}}{12.92} \text{ else } \mathcal{B} = \left(\frac{\text{B} + 0.055}{1.055}\right)^{2.4}$$

In the sRGB colour space—R, G, B range from 0.0 to 1.0.

Figure 5.1 shows the contrast ratio between the foreground and background colours. A Contrast ratio less than about 5 means text is becoming difficult to read.

Exercise 5.4: Print Figure 5.1 on a colour ink-jet printer, on a colour laser printer, on a monochrome laser-printer, on different medium—What are the differences in readability?

How do the printed versions compare with the screen version?

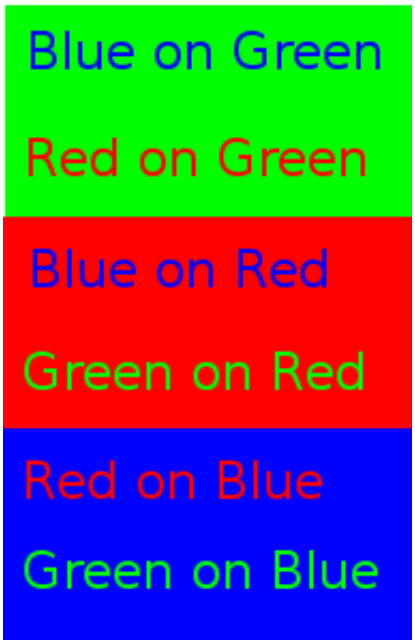
Web page designers an avoid vision issues for users if they follow a few simple rules—

Avoid using foreground and background colours of similar hue Users with vision that is somewhat colour deficient are often unable to differentiate between colours of similar hue when those colours are also of similar lightness and saturation. See Figure 5.2.

There are also other colours and colour combinations to avoid. Assuming equal saturation and lightness colours to be aware of are (see Figure 5.3)

- blue foregrounds—the eye cannot focus on blue very well. The human eye has far less sensitivity to blue than any other colour.
- since the eye is less sensitive to blue—blue backgrounds with a contrasting foreground (yellow for instance) can be very easily read and not a strain on the eyes.
- the eye cannot focus on shades of red and blue simultaneously so red/blue combinations of foreground/background colours should be avoided.

Figure 5.3: Comparison of Red, Green and Blue foreground and background combinations.



Avoid using foreground and background colours of similar saturation Colours with low saturation—that is, colours close to achromatic grey— can be used as backgrounds, and can reduce eye-strain by reducing the glare from screens—but must have a foreground that does not have a similar hue, saturation or lightness.

Avoid using foreground and background colours of similar lightness—keep contrast high Dark text on dark backgrounds or light text on light backgrounds may not be readable on all monitors or by all people who visit the site. Yellow and black are examples of contrasting colours—this is why they are used on traffic signs throughout the world.

Avoid using busy background tiles Tiled background images with patterns of multiple hues, saturations or lightness levels will make text extremely difficult to read. Never use speckled or textured backgrounds.

Make important items' colours distinguishable in two ways. To make pages readable to all users foreground and background colours should be significantly different in two variables (hue and lightness, for example).

Exercise 5.5: Print Figure 5.2 on a colour ink-jet printer, on a colour laser printer, on a monochrome laser-printer, on different medium—What are the differences in readability?

How do the printed versions compare with the screen version?

Memory

Memory is critical to user being able to utilise a site. If users are unable to remember anything about the structure of a site as they browse it, they will become hopelessly lost—and are unlikely to return. Users tend to employ a simple maxim: Minimise effort and Maximise gain! A user will not spend a great deal of time understanding a site unless the outcome warrants the expenditure of effort—users are "lazy"!

A few simple ideas can be employed to ensure a visitor to a site is not presented with too complex a site that means they will avoid it—

Recognition is easier than recall There are plenty of examples how recognition is easier than recall. Most people consider multiple-choice tests easier than fill-in tests—the difference between recognising the correct answer and recalling the correct answer. If links are not to be coloured on a site then links will never look as though they have been visited forcing users to recall where they have been. If links change colour after they have been visited, then a user only has to recognise the different colour for a visited link.

Only three actions can be memorised sequentially Web users appear to be able to remember about the last three sequential pages as they browse (consider how many instructions you can remember when asking for directions!). As a user jumps through a myriad of pages some of the pages will be memorable—mainly because they are different—for instance site home pages. However, if you wish users to remember a path through your site then they tend to remember only three pages sequentially. Therefore you cannot expect users to remember a sequence or path longer than three items without repeated use—this implies that all content should be approximately three clicks away from the home page!

Response Times

Web users generally, are more patient with a site if they are unfamiliar with it. Sites that could be considered single visit-sites, such as promotional sites for products (movie sites, computer equipment sites, designer portfolio sites etc.), can afford huge download times. The user will wait for the animation or multiple images because they want information on the products being displayed. On subsequent visits though, patience for the glitz wears thin. The needs and desires of the first-time visitor, are different than the frequent visitor to a site.

The amount of time a user will wait will vary based on the individual user, and the potential benefit gained from waiting. However, there are some things that can be said about the elapsed time a user is willing to wait—

0.1 Seconds An event on this times scale appears almost instantaneous to the user.

1.0 Seconds The user is relatively engaged with the event and not easily distracted from what is happening on the screen

10 Seconds This is the suggested limit for keeping the user's attention focused on the page. Even with a "progress bar" many users will become bored and move on to something else.

>10 Seconds The user will either leave the site or open up new tabs in the browser and visit other sites while waiting for the download to complete.

Stimulus

Users are constantly being bombarded by stimuli from sites. The text, the links, the graphics, the animations, the sound, ... all create a cacophony of stimuli that the user tries to distill useful meaning from—useful meaning to the user, that is! The three primary ways people filter data is via setting thresholds, concentrating on a part of the whole, and sensory adaption.

Thresholds Instead of dealing with minute differences between objects users tend to notice differences only when they exceed a particular threshold. Thresholds suggest making objects or pages noticeably different from each other so that users will be easily able to understand their difference. Designers should not force their users to spend time and effort trying to interpret differences between objects on a page, since it is frustrating and takes time and concentration away from the main goal of getting the user to read the page or perform a task. For example, consider if link and text colour on a page are too similar—the user will spend time carefully inspecting text for underlines instead of reading the content.

Concentrating on a Subset Users invariably focus on only a part of a page at any given moment. Concentrating on the information in that section with the rest of the page filtered out. A good site has lots of choices but provides the visitor with the ability to focus on what they are looking for. One way to do this is by grouping similar items together and separating groups of items with neutral space. Break text up with subheadings, lists, short paragraphs, highlighted keywords etc. A site design should strive to limit competing objects on a page.

Sensory Adaption Sensory adaption occurs when users become so used to a particular stimulus that they no longer respond to it—at least not consciously. Sensory adaption suggests that numerous fonts, banners, animations, and coloured regions on a page may go unnoticed over time. A user's full attention can be "grabbed" by presenting them with something unexpected (such as a pop-up window)—unfortunately disturbing a user's focus on the task at hand can make them feel uncomfortable because of the lack of consistency, and annoy them enough so they leave.

Movement

Web sites are generally manipulated using the keyboard or the mouse—therefore a good web site design should attempt to minimise the user's need to use these devices. Few sites consider that users may prefer using the `TAB` key, instead of the mouse, to move through choices on a page. Though data entry pages (forms pages) may be optimised for quick navigation via the keyboard, non-form pages rarely are.

Consideration of the work users perform moving their mouse around the screen also needs to be taken into account. Moving the pointer around the screen takes effort and time, a button or link press may take up to a few seconds if a user has to move the pointer a long distance or focus on clicking a very small button. A few simple rules can be applied to improve the users speed of use:

Minimise mouse travel distance between successive choices The page design should be such that when a user selects a link from a navigation list say, the new page will appear with the navigation list under the pointer. The user will then not have to move the mouse far for another selection. This requires consistency of design from page to page and also assumes that the user will be navigating through the site exclusively using the provided navigation tools. This is not the case.

Place navigation tools near the Back/Forward buttons User's will also navigate through a site using the Web browser's Back/Forward buttons. Minimising mouse travel to the Back/Forward buttons means placing the primary navigation tools near the browser buttons. For most browsers this means near the upper left of the page—this of course also places the primary navigation tools away from the scroll bars—another section of the browser window heavily used! But usability also suggests that pages should not be so long that scrolling is necessary!

Exercise 5.6: What are your Web browser's keyboard commands? How do you navigate through a site using the keyboard alone?

The `TAB` key is used to jump from one data entry field or hypertext link to another—how many sites can you find that have clearly been designed so the `TAB` key is effective?

Make clickable regions large enough to be easily selected Clickable regions should be large enough for users to move to them quickly and select them accurately. The further the region is away from the main focus of the page the larger it will have to be. This means—

- enough text should be part of an anchor to make it unambiguously a link,
- images should be large enough to be noticeable and easily clickable, and

- all clickable regions (text and images) should be spaced from each other so that adjacent choices are not accidentally selected!

5.2.3 Web Conventions

Most users will expect Web sites to follow a standard design—the common interface conventions established by the Web's most heavily used sites. Users will spend more of their time on other sites than yours—that is where users form their expectations of how the Web works! A site design should keep users expectations in mind.

Some common web conventions are:

- **Banner across the top with left corner logo.** Users expect the site banner to appear across the top of the page with a site logo in the upper-left corner. This should be repeated on every page.
- **Sectional navigation on left or under banner.** The main navigational components of a site are expected to be either vertically down the left side of the page or horizontally across the top of the page under the banner. The chosen sectional navigational form is expected to be repeated on all pages.
- **Logo is a link to site home page.** The site logo is expected to be an anchor that will return the user to the site's home page.
- **Text links are repeated at the bottom of the page.** Sites tend to repeat textual navigation at the bottom of the page, particularly if the top or side sectional navigation is graphical.
- **Back-to-top link at the bottom of long pages** A back-to-top anchor is generally included at the bottom of the page to quickly jump the user to the top. This is particularly noticeable on long pages that may not be expected to fit on a single screen.
- **Special print style for heavily printed pages** Heavily printed pages are provided with printer-friendly versions of the site style file.
- **Shopping cart in the upper right** Typically, the shopping cart links are found in the upper-right corner of the page.
- **Anchors are blue and underlined** Most text hyper-links are blue and underlined. A different colour for links will just confuse most users.
- **Search and site map in addition to sectional navigation** Large sites will provide a key-words search facility—normally signified with the word "Search". Most sites will also have variations on a site-map page that will have links to the site sections in more detail than the main sectional navigation tools.

5.3 Accessibility

Providing accessibility for people who may have deficiencies involving sight, hearing, or other physical capabilities is not just a nice idea—it

is actually required in some countries and for some organisations (such as government agencies)—many companies could be sued if they do not account for all users (see Maguire v. SOCOG 2000, in the list of readings below).

Making a Web site accessible is something that should be done, not just because of a local law or to avoid litigation, but because doing so will result in a much better Web site for all users. Creating systems that are accessible to all users also creates benefits for all users, regardless of capability. Consider "audio books", initially considered for the blind—now used by everyone, "curb cutouts" originally made for wheelchairs, make crossing a street easier for all.

The Web Accessibility Initiative (WAI) of the World Wide Consortium (W3C) is not only concerned with creating sites that are accessible to people with disabilities, but also making sites that are accessible by anyone who might be operating in a different environment than is considered "the norm" by the site designer. A site designer should always consider that users may have different operating constraints. From the WAI consider the following list of potential user constraints:

- users may not be able to see, hear, or move easily, or not be able to process some types of information easily, or at all.
- users may have difficulty reading or comprehending text because of language problems.
- users may not be able to use a keyboard or mouse because of access method (a mobile phone for example) or physical disability.
- users may be operating in a less than ideal environment, such as a text-only screen, a small screen, a monochrome screen, or a slow Internet connection.
- users may be accessing the site in a non-standard environment where they may be affected by environmental factors—accessing the Web in an airport, factory floor, from a car, or out of doors in direct sunlight .
- users may have an older browser or a non-standard browser or operating system or use an alternative form of user interface, such as voice access.

To deal with these issues the W3C has issued suggestions to improve the accessibility of a site. These suggestions help make a site accessible for everyone—whether they have a disability or not. The W3C suggestions are:

Ensure that documents are clear and simple An obvious but important suggestion—simplicity will lead to greater accessibility. Not everyone will be able to read a language well, usability is directly related to simplicity and consistency.

Provide context information Pages should be designed so that the purpose of every element on a page is clear. For example, the meaning of links should be clear through the use of tool-tips (the

`title` attribute), forms should be designed so that what is required is clear.

Provide clear navigation mechanisms A site should provide basic navigation that is easy to understand and at a consistent location on the page. Navigation aids such as search engines, site maps, and site indexes should also be provided.

Use W3C technologies and guidelines The site designer should always try and follow W3C specifications and guidelines. Be aware though, of the specifications and versions that are supported by the major browsers.

Design for device independence Web sites should be designed so that they can work on different devices, including those with different screen sizes, different viewing devices (mobile phones to wide screens), and different manipulation devices (touch screen, keyboard and/or mouse)

Use markup and style sheets, and do so properly Web pages should use W3C defined (X)HTML for structure and CSS for presentation. Proprietary markup or presentation elements should be avoided, as well as technologies that may not be rendered consistently on different browsers.

Clarify natural language usage The predefined logical markup elements in (X)HTML should be used to indicate acronyms, definitions, quotations, etc. Also the document's language should be clearly indicated within a document to allow a browser to switch (if it is able) to another language.

Create tables that transform gracefully Tables should never be used for page layout—they are designed for presenting tabular data. When a table is used it should be provided with a clear summary of the contents, a clear caption, column and/or row headings, and other indicators of the meaning of cell contents.

Don't rely on colour alone Not every user will be able to view colours properly. In general, colour combinations with similar hue or contrast should be avoided—particularly if they are likely to be viewed on monochrome displays or by people with colour vision deficits. If colour alone is used to convey information, such as what constitutes a link, people who cannot differentiate between certain colours and users with non-visual or monochrome displays will not be able to discern what is being presented.

Provide equivalent alternatives to auditory and visual content Web sites should not rely only on one form of communication.

If audio is used, a text transcript of the message should also be supplied. If image buttons are used as links, text links should also be provided.

Ensure pages the feature new technologies transform gracefully Any new technologies used on a page must degrade gracefully under older browsers, older plug-ins or with the feature turned off in the browser. For example, if a page relies on Javascript will the page work with it turned off? Or at least will the page fail gracefully?

Ensure user control of time-sensitive content changes Any moving, blinking, scrolling or auto-updating objects or pages (if they must be used) must be able to be paused or stopped by the user. Auto-updating objects on a page are highly annoying, but more importantly they will make it difficult for a user to focus on the site.

5.4 Usability Guidelines

The following lists have been gleaned from http://www.useit.com/, a site that at least attempts to collect quantitative data for most of its suggestions. You will note that a number of ideas are repeated—this would tend to suggest they are fundamental to the Web and are often ignored!

5.4.1 Ten Good Design Ideas

This is a list of ideas that if adopted in a site design will increase the usability of the site—

(a) **Name and Logo on every page:** Place the site name and logo on every page. the logo should be a link to the site home page—but not on the home page itself. Never have a link that points directly back to the current page.

(b) **Provide a Search facility** If the site is large, more than 50–100 pages. The search facility should be clearly labelled with the word "Search".

(c) **Simple headlines and page titles** Write straightforward and simple headlines and pages titles that clearly explain what the page is about and will make sense when read out-of-context in a search engine's result listing.

(d) **Facilitate page scanning** Users do not read in detail Web pages they scan for the information they want. Structure the page to facilitate scanning and help users ignore large parts of the page in a single glance. For example, use grouping and subheadings to break long list into several smaller units.

(e) **Structure the content space** Do not cram everything about a topic into a single page—use hyper-text to structure the content into a starting page that provides an overview and several secondary pages that each focus on a specific topic. The goal is

to allow users to avoid wasting time on those subtopics they are not interested in.

(f) **Use link titles** Use the `title` element to provide users with a preview of where each link will take them, *before* they have clicked on it.

(g) **Do the same as everyone else** Design your site to follow the design lead of heavily visited Web sites—users expect all sites to work similarly.

(h) **Accessibility** Ensure that all important pages are accessible for uses with disabilities, especially blind users.

(i) **Avoid cluttered image pages** Avoid cluttered or bloated image pages with lots of photos—for example, product photo pages. Instead have small images on individual topic or product pages and link the image to one or more larger images with as much detail as the user needs. This will vary depending on the purpose of the pages. Images that might require advanced features such as being zoomable or rotatable should be reserved for the secondary pages.

(j) **Use relevant image reduction** When creating a thumbnail version of an image do not simply resize the original image so that it is indecipherable—zoom in on the relevant detail and use a combination of cropping and resizing.

5.4.2 Ten Bad Design Ideas

This is a flexible list that changes all the time as site designers are continually coming up with bad Web design habits—

(a) **Bad Search Engine** Search is a user's lifeline when navigation fails. Overly literal search engines reduce usability in that they are unable to handle typographic errors, plurals, etc. Even though advanced search can sometimes help, simple search works best, and search should be presented as a simple text input box with the word "Search" beside it, since that is what users are looking for.

(b) **Not changing the colour of visited links** A good understanding of past navigation helps users understand their current location. Knowing their past and present locations will help users decide where to go next. More importantly, knowing which pages they have already visited frees users from unintentionally revisiting the same pages over and over again. These benefits are only available under one important assumption: that users can tell the difference between visited and unvisited links because they are in different colours! When visited links don't change colour, users exhibit increased navigational disorientation in usability testing and unintentionally revisit the same pages repeatedly.

(c) **Non-scannable text** A wall of text is deadly for an interactive experience—interactive users want to scan pages for the information they are seeking—not read them word for word.

Web pages should be written and formatted for online content not for print content. To support scannability, use well-documented formating techniques—

- sub-headings
- bulleted lists
- highlighted keywords
- short paragraphs
- Structure the content in the "inverted pyramid" style. Also called the "summary news lead" style. In this style the most substantial, interesting, and important information the writer means to convey is at the start of the text, other material follows in order of diminishing importance.

  A good style for Web page scanning as readers can leave the text at any point and understand it, even if they don't have all the details.
- a simple writing style—avoid hyperbole.

(d) **Fixed font size** CSS style sheets give Web site designers the power to fix font sizes. About 95% of the time, this fixed size is *tiny*, reducing readability significantly for most people over the age of 40.

Respect the user's preferences and let them resize text as needed. Also, always specify font sizes in *relative* terms—not as an *absolute* number of points or pixels.

(e) **Violating design conventions** Consistency is one of the most powerful usability principles: when things always behave the same, users don't have to worry about what will happen.

The more users' expectations are proven right, the more they will feel in control of the system and the more they will like it. And the more the system breaks users' expectations, the more they will feel insecure.

(f) **Looking like an advertisement** Selective attention is powerful, and Web users have learnt to stop paying attention to any advertisements that get in the way of their goal-driven navigation. Therefore, it is best to avoid any designs that may resemble advertisements. An ever changing field, but currently avoid—

- **Banners** Users never give their attention to anything that looks like a banner advertisement.
- **Animation** Users ignore areas with blinking or flashing text or other aggressive animation.
- **Pop-up Windows** Users close small pop-up windows before they have even fully rendered.

(g) **Opening new browser windows** Opening new browser window has a number of negative consequences—

- Users can view it as a hostile act—since the site is taking over the user's machine.
- It disables the "Back" button which is the normal way users return to previous sites.
- If they are using a small monitor where the windows are maximised to fill up the screen—they may not even notice the that a new window has opened.
- Users hate unwarranted pop-up windows. When they want the destination to appear in a new page, they can use their browser to do it – assuming, of course, that the link is not a piece of code that interferes with the browser's standard behaviour.

(h) **Not answering user's questions** Users are highly goal-driven on the Web. They visit sites because there is something they want to accomplish. The ultimate failure of a Web site is to fail to provide the information users are looking for!

Sometimes the answer is simply not there and they leave the site. Other times the specifics are buried under a thick layer of hyperbole. Since users don't have time to read everything, such hidden information might almost as well not be there.

(i) **Page titles with low search engine visibility** Search is the most important way users discover websites. The page title, contained within the XHTML `<title>` tag is almost always used as the clickable headline for listings on search engine result pages. Search engines typically show the first 60 characters of the page title—so they need to be concise! The page title is the main tool for attracting new visitors from search listings.

Page titles are also used as the default entry when users bookmark a site.

(j) **Non-HTML documents for online reading** Users hate coming across a non-HTML file while browsing, because it breaks their flow. PDF or Microsoft Office Documents are great for printing and for distributing manuals and other big documents that need to be printed. Reserve it for this purpose and convert any information that needs to be browsed or read on the screen into XHTML pages.

5.5 Questions

Short Answer Questions

Q. 5.7: What do we mean when we suggest that good design is "user-centred"?

Q. 5.8: "Usability" is a term used a lot in this chapter—what does it mean in the context of Web design?

Q. 5.9: If users do not read Web pages in detail but only scan web pages looking for the information they want—how would you design a page to facilitate scanning?

Q. 5.10: Why is it a good idea for All users to follow "Accessibility" guidelines?

Q. 5.11: Why is it a good idea to use the `title` attribute with the anchor element?

Q. 5.12: Why isn't it a good idea to use red and blue hues as foreground and background colours?

Q. 5.13: Explain why that every page on a site should be at most 3 "clicks" from the home page?

Q. 5.14: Explain why it is a good idea when designing a web site to follow the W3C standards and guides?

Q. 5.15: Why is it not a good idea to use pop up windows on a site?

Q. 5.16: Why should the title of a page—the text found in the `title` container—be simple, short and concise?

Q. 5.17: What is the "inverted pyramid" style of structuring text? Why is it a good idea to use it when writing for the web?

Q. 5.18: Why should you never use a fixed font size?

5.6 Further Reading and References

- `www.usabilitynet.org` — A European Union site to promote usability and user-centred design.
- Design guidelines for the Web can be found at `http://www.usabilitynet.org/tools/webdesign.htm`.
- Accessibility guidelines for the web can be found at `http://www.usabilitynet.org/tools/accessibility.htm`.
- http://www.useit.com/ — Jakob Nielson's Web site, a leading expert on Web usability.
- www.useit.com/homepageusability/guidelines.html — 113 guidelines for ensuring homepage usability.
- http://www.w3.org/WAI/WCAG20/quickref/ — W3C Accessibility quick reference.
- Maguire v. SOCOG 2000 available on Wikipedia.

Department of Mathematics and Computing, USQ.
(This file created: June 12, 2012)

Chapter 6 PHP: Hypertext Preprocessor

PHP[1] was created by Rasmus Lerdorf in June 1995, to make various common web programming tasks easier and less repetitive. The goal of that release was to minimise the amount of code required to achieve results, this led to a design decision that PHP code would be embedded inside HTML pages.

Embedded PHP commands within a HTML document means that the HTML document goes from being a static document that is read by the server from the document tree to a dynamic document that is parsed[2] and executed by the PHP interpreter before being sent to the web client.

Chapter contents

[1] The acronym PHP stands for "PHP: Hypertext Preprocessor" which is known as a *recursive* acronym

[2] *Parsing* is the process analysing a sequence of tokens (or words) to determine their grammatical structure.

6.1 Syntax

The PHP interpreter assumes everything is plain text to be ignored. The advantage of this is that PHP commands can be embedded into a HTML document. This method of parsing means that the PHP elements of a script are "islands of code" that are interpreted independently of the surrounding "sea" of HTML. These "islands of PHP code" are not independent of each other though—they are parsed as if they where one contiguous script.

A PHP scripting block starts with `<?php` and ends with `?>`. A PHP scripting block can be placed anywhere within a text document[3].

Script 6.1: A basic PHP script

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>Minimilist HTML &amp; PHP document</title>
</head>
<body>
  <?php echo "Hello World"; ?>
</body>
</html>
```

Each PHP statement must end with a semicolon. The semicolon is a separator used to distinguish one statement from another.

When interpreting PHP via a web server—the server passes the file to the PHP interpreter[4]—which executes the PHP code in the file—the output of the code becomes a part of the document—which is then sent to the web client.

The server recognises a file that contains PHP code by the MIME type of the document (see Chapter 8. The default MIME type for PHP

[3] Within XML documents the block defined by `<?name` and `?>` is called a Processing Instruction. The "name" defines the interpreter the instruction is meant for

[4] In Apache the PHP interpreter is incorporated into the server as a server module. The server compiled for this course has PHP built into it.

documents is `application/x-httpd-php` which is normally mapped to files that end in `.php` or `.phtml`.

Script 6.1 shows a basic PHP script that will print out "Hello World!" to become part of a basic HTML page.

Exercise 6.1: Copy the code of Script 6.1 to file called `script01.php` and a file `script01.html`. Serve the files from the Apache server. Explain the differences in the displayed pages.

Exercise 6.2: Copy the code of Script 6.1 to file called `script01.phps`. How is this page displayed?

PHP has a built in source displayer.

Exercise 6.3: Create a file called `info.php` and place the following PHP code in it—

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>PHP Interpreter</title>
</head>
<body>
  <?php phpinfo(); ?>
</body>
</html>
```

What is the output when this resource is downloaded from your web site?

6.1.1 Comments

PHP recognises C, C++ and scripting style comments.

The comment styles '`//`' or '`#`' only comment to the end of the line or the current block of PHP code, whichever comes first. This means that HTML code after `// ... ?>` or `# ... ?>` will be printed—`?>` breaks out of PHP mode and returns to HTML mode, and `//` or `#` will not influence that (as long as `?>` has a space in front of it!).

6.2 Variables

Variables in PHP start with a dollar sign followed by the name of the variable. The variable name is case-sensitive. A valid variable name starts with a letter or underscore, followed by any number of letters, numbers, or underscores.

6.2.1 Types

PHP has eight data types— integer, float, boolean, string, array, object, resource and null.

Script 6.2: Including comments in PHP scripts

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>PHP Comments</title>
</head>
<body>
  <p><?php echo "Hash is a comment in PHP";
    # Comments cannot effect
    # the end PHP construct ?> </p>

  <p><?php echo "Double slash is a comment in PHP";
    // Comments cannot effect
    // the end PHP construct  ?> </p>
  <p><?php echo "C style block comments can also be used";
    /* C style block comments are different
       in that the comment covers multiple lines
       and must be terminated */  ?> </p>
</body>
</html>
```

integer Integers are whole numbers, either positive or negative.

float Floats hold fractional numbers as well as very large integer numbers,

boolean Booleans hold either true or false. Internally to PHP, booleans are just integers. PHP considers the number 0 to be false, and everything else to be true.

string Strings hold characters—literally "a string of characters". Strings can be as short or as long as you want, there's no limit to size. PHP considers strings to be case-sensitive which means that some string functions have case-insensitive equivalents.

array Arrays hold multiple values like a container, and can even hold arrays of arrays (*multidimensional arrays*).

object Like arrays, objects are complex variables that have multiple values, but they can also have their own functions (called *methods*) associated with them. PHP 5 allows for object-oriented programming. Object-oriented programming using PHP will not be covered in this course.

resource PHP provides an interface into many libraries that deal with very different sorts of data. Resources are anything that is not

Script 6.3: Using variables in PHP scripts

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>PHP Variables</title>
</head>
<body>
  <?php
       $txt = "Hello World!";
       echo $txt;
       $num = 12379;
       echo $num;
  ?>
</body>
</html>
```

PHP data. Internally, a resource variable holds a handle (pointer) to the actual data, because it is created outside of PHP.

null Null is a special type and represents a variable with no value.

Loosely Typed

In PHP a variable does not need to be declared before being set. In Script 6.3 variables where declared and a value assigned in the same line. PHP automatically converts the variable to the correct data type, depending on how it was set.

In strongly typed programming languages (C or C++ for example) the name and type of a variable must be declared before it can be used. Most scripting languages are loosely typed (perl, python or shell scripts for example).

In PHP the variable is declared automatically when you use it. Unfortunately this has the problem that PHP will issue warning messages if variables are used before they have a value assigned to them.

Unset variables have the special type NULL.

Scope

Each variable has a life span in which it exists, known as its scope. It is technically possible for a PHP script to have several variables called `$v` in existence at one point in time; however, there can only be one active `$v` at any one time.

Any variables not set inside a function or an object are considered global. That is, they are accessible from anywhere else in the script, except inside another function or object.

6.2.2 True or False

The constant "true" is TRUE, and "false" is FALSE

PHP considers some values to be equivalent to TRUE, and others equivalent to FALSE. Most non-zero numbers are true (e.g., 1, 73, 345129876), but 0, 0.0, 0.00000000 are all FALSE.

Nearly any string with a value in it is considered to be true, so "k", "-234", "false", and "a string of characters" are all TRUE. However, an empty string "" and "0" are both FALSE. Confusingly, the string "0.0" is TRUE.

An empty array as with an empty string are considered FALSE.

The special type NULL is considered FALSE.

6.2.3 Strings

A string can be specified in three ways—single quotes, double quotes or *heredoc* syntax.

Single Quotes The simplest way to specify a string is to enclose it in single quotes (see Script 6.4).

To include single quote, you will need to escape it with a backslash (\) (see Script 6.4).

Script 6.4: Using single quotes to define a string

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>Using single quotes</title>
</head>
<body>
<?php
     $str1 = '<h3>Single quoted string</h3>';

#  Single quotes must be escaped within
#  strings defined by single quotes
     $str2 = '<h3>Don\'t do that</h3>';

     echo $str1;
     echo $str2;
?>
</body>
</html>
```

Double Quotes If the string is enclosed in double-quotes ("), PHP understands more escape sequences and PHP variables will be parsed. Table 6.1 list the recognised escape characters.

Table 6.1: Escaped Characters

Escaped Character	Meaning
\n	linefeed (LF or 0x0A (10) in ASCII)
\r	carriage return (CR or 0x0D (13) in ASCII)
\t	horizontal tab (HT or 0x09 (9) in ASCII)
\\	backslash
\$	dollar sign
\"	double-quote
\[0-7]{1,3}	a character in octal notation
\x[0-9A-Fa-f]{1,2}	a character in hexadecimal notation

The most important feature of double-quoted strings is the fact that variable names will be expanded. To correctly parse variable names in a string the variable name should be enclosed in braces (see Script 6.5).

Script 6.5: Using double quotes to define a string

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>Using double quotes</title>
</head>
<body>
<?php
     $name = 'John M. Smith';

     echo "<h3>Double quoted string</h3>";
# place braces around variables
# ensure the interpreter gets it right!
     $str2 = "<h3>&ldquo;{$name}&rdquo; owes me \$30</h3>";

     echo $str2;
?>
</body>
</html>
```

Heredoc Another way to delimit strings is by using heredoc syntax. Heredoc syntax are a way of defining a multi-line string. Heredoc syntax defines a unique word that terminates the multi-line string (see Script 6.6).

It is very important to note that the line with the closing word contains no other characters, except possibly a semicolon (;). That means especially that the identifier may not be indented, and there may not be any spaces or tabs after or before the semicolon.

Script 6.6: Using *heredoc* syntax to define multi-line strings

```
<?xml version="1.0" encoding="UTF-8"?>
<?php
$heading='HereDoc Example';
$htmltype = <<<EOT
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
EOT;
$page = <<<EOT
{$htmltype}
<head>
 <title>Using HereDoc Syntax</title>
</head>
<body>
<h1>{$heading}</h1>
This page is produced using Heredoc syntax.
</body>
</html>
EOT;
echo $page;
?>
```

Exercise 6.4: In Script 6.6 the variables in the `$page` heredoc are surrounded by braces why? Why does nothing happen if the braces are removed?

Experiment with strings in PHP and explain when braces are required around variables within strings.

Access By Character

Characters within strings may be accessed and modified by specifying the zero-based offset to the desired character. For example,

```
$str[42]='6'
```

will modify the 43rd character in the string—string indexing starts at zero.

String Functions

The ability to manipulate strings is important when building web pages with PHP. For this reason PHP has a rich collection of string functions and a powerful Perl-like regular expression engine.

Table 6.9 (in §6.9) lists most, but not all, of the string function that are built into PHP.

Regular Expressions

Regular expressions offer more power over strings than string functions, but can be difficult to use because their syntax can be confusing.

Regular expressions can:

- Replace text
- Test for a pattern within a string
- Extract a substring from within a string

A regular expression describes a pattern or a sequence of characters. An *expression* is not something to be interpreted literally—t is something that needs to be evaluated.

The syntax of a regular expression must first be evaluated (the regular expression is in effect compiled) to produce a pattern. The pattern is then compared against a string (for example a line of text). To see if the string contains a matching substring the first character of the pattern is compared to the first character of the string. If there is a match the second character of the pattern is compared to the second character of the string. If the current pattern character fails to match the current string character then the matching starts again at the start of the pattern but now starting one character along in the string.

The power of regular expressions comes from the fact that patterns are not restricted to literal characters. Regular expressions define a set of `meta`-characters. Each meta-character has a special meaning in a regular expression, that is, it can represent characters other than itself.

PHP contains two ways to perform regular expressions, known as POSIX-extended (equivalent to Unix Extended Regular Expressions) and Perl-Compatible Regular Expressions (PCRE) . The PCRE functions are more powerful and faster than the POSIX ones—and also incorporate the POSIX-extended regular expression syntax as a subset.

The functions and syntax for regular expression matching are beyond the scope of this study-book and will not be covered—but can be found in the PHP documentation.

6.2.4 Arrays

PHP has built-in support for arrays of data, and they are created using the `array(...)` function or using the special operator `[...]`.

An array is a normal PHP variable, but it works like a container. It can contain values of any type. Values within an array are accessed using a *key*. Each *key:value* pair is called an element of the array.

Exercise 6.5: Add comments to each line of Script 6.7.

Exercise 6.6: Explain the difference between an *indexed* array and an *associative* array. Use Script 6.7 for examples.

Exercise 6.7: Modify Script 6.7 to use the array function `array_push()` to add more fruit to the `fruit` array.

At what array positions does `array_push` add the fruit.

Also use `array_unshift()`—how is it different to `array_push()`.

Explain the purpose of the functions `array_pop()` and `array_shift()`

A key may be either an integer or a string. If only the value is specified, then the key is assumed to be an integer starting at zero for the first element. This produces the traditional integer-indexed array. If the key and the value are both specified then you have a traditional associative array. See Script 6.7 for examples.

6.3 Operators

An operator is something that you feed with one or more values or expressions, and which yields another value.

PHP has three groups of operators—the *unary* operator which operates on only one value, the *binary* operator which operates on two values (by far the largest group) and the *ternary* operator which operates on three values (there is only one operator in this group).

6.3.1 Arithmetic Operators

Table 6.2 lists the arithmetic operators found in PHP.

The division operator ("`/`") returns a float value every time, even if the two operands are integers (or strings converted to integers).

The Modulus operator (%) returns a negative value for a negative numerator.

Table 6.2: Arithmetic Operators

Example	Name	Result
`-$a`	Negation	Negative of `$a`.
`$a + $b`	Addition	Sum of `$a` and `$b`.
`$a - $b`	Subtraction	Difference of `$a` and `$b`.
`$a * $b`	Multiplication	Product of `$a` and `$b`.
`$a / $b`	Division	Quotient of `$a` and `$b`.
`$a % $b`	Modulus	Remainder of `$a` divided by `$b`.

All of the binary operators above can be combined with the assignment operator ("`=`") to produce a shorthand binary operator—

```
$a = 5;
$b = 5;
$a *= 7;
$b = $b * 7;
```

both `$a` and `$b` have the value `35`. The statements are equivalent.

Exercise 6.8: What is the return value of the quotient `/` operater if one of the variables contains a string?

What does it mean to multiply two strings together?

6.3.2 Assignment Operator

The basic assignment operator is "=". This operator is *not* that same as "equal to". It means that the left operand gets set to the value of the expression on the right.

The value of an assignment expression is the value assigned. That is, the value of "`$a = 3`" is 3. This means that the following—

```
$a = ($b = 5) * 9;
```

assigns the value `5` to `$b` and `45` to `$a`.

String Operator

There are only two string operators in PHP—

Concatenation—returns the second value appended to the first:

```
$c = $a . $b
```

Shorthand concatenation—appends the second value to the first:

```
$a .= $b
```

6.3.3 Comparison Operators

Comparison operators allow you to compare two values. The return value is either TRUE or FALSE. Table 6.3 lists

Table 6.3: Comparison Operators

Example	Name	Result
`$a == $b`	Equal	TRUE if `$a` is equal to `$b`.
`$a === $b`	Identical	TRUE if `$a` is equal to `$b`, and they are of the same type.
`$a != $b`	Not equal	TRUE if `$a` is not equal to `$b`.
`$a <> $b`	Not equal	TRUE if `$a` is not equal to `$b`.
`$a !== $b`	Not identical	TRUE if `$a` is not equal to `$b`, or they are not of the same type.
`$a < $b`	Less than	TRUE if `$a` is strictly less than `$b`.
`$a > $b`	Greater than	TRUE if `$a` is strictly greater than `$b`.
`$a <= $b`	Less than or equal to	TRUE if `$a` is less than or equal to `$b`.
`$a >= $b`	Greater than or equal to	TRUE if `$a` is greater than or equal to `$b`.

Unlike strongly typed languages—weakly typed languages allow you to compare different types. Normally one or both of the operands are converted so they can be compared.

Table 6.4 shows how different types are compared.

Table 6.4: Comparison of different types (order is unimportant)

Operand A	Operand B	Result
null	**string**	Convert **null** to "". Attempt numerical comparison otherwise lexical comparison.
string	**string**	Attempt numerical comparison otherwise lexical comparison.
boolean	anything	Convert to booleans: FALSE<TRUE
null	anything	Convert to booleans: FALSE<TRUE
string	**number**	Convert string to number—then standard comparison.
array	**array**	Array with fewer elements is smaller. If they are the same length compare value by value—until different values found. Same length arrays must have the same keys—otherwise they are incomparable.

6.3.4 Ternary Operator

Another conditional operator is the "? :" (or ternary) operator. It is the same as the operator found in C.

The expression

(expr1) ? (expr2) : (expr3)

evaluates to *expr2* if *expr1* evaluates to TRUE, and *expr3* if *expr1* evaluates to FALSE.

See Script 6.8 for examples of using the ternary operator.

6.3.5 Increment/Decrement Operators

PHP supports C-style pre- and post-increment and decrement operators. See Table 6.5.

Table 6.5: Increment/Decrement Operators

Example	Name	Result
`++$a`	Pre-increment	Increments `$a` by one, then returns `$a`.
`$a++`	Post-increment	Returns `$a`, then increments `$a` by one.
`--$a`	Pre-decrement	Decrements `$a` by one, then returns `$a`.
`$a--`	Post-decrement	Returns `$a`, then decrements `$a` by one.

6.3.6 Logical Operators

Table 6.6 lists the logical operators.

Table 6.6: Logical Operators

Example	Name	Result
`$a and $b`	And	TRUE if both `$a` and `$b` are TRUE.
`$a or $b`	Or	TRUE if either `$a` or `$b` is TRUE.
`$a xor $b`	Xor	TRUE if either `$a` or `$b` is TRUE, but not both.
`! $a`	Not	TRUE if `$a` is not TRUE.
`$a && $b`	And	TRUE if both `$a` and `$b` are TRUE.
`$a || $b`	Or	TRUE if either `$a` or `$b` is TRUE.

The reason for the two different variations of "and" and "or" operators is that they operate at different precedences. See Table 6.8 for operator precedence.

As in many languages logical expressions are evaluated left to right and evaluation stops when an unambiguous answer is assured. Script 6.8 illustrates this ability to short-circuit evaluation.

Exercise 6.9: Comment every line of Script 6.8.

Exercise 6.10: Explain the following line of code

```
$fh=fopen("filename.txt","r") or
          exit("Failed to open file");
```

6.3.7 Array Operators

Table 6.7 lists the array operators. See also Table 6.10 for array functions.

The Union operator (+) appends the right hand array to the left hand array—without over-writing values in the left hand array if keys are duplicated.

Table 6.7: Array Operators

Example	Name	Result
`$a + $b`	Union	Union of `$a` and `$b`.
`$a == $b`	Equality	TRUE if `$a` and `$b` have the same key/value pairs.
`$a === $b`	Identity	TRUE if `$a` and `$b` have the same key/value pairs in the same order and of the same types.
`$a != $b`	Inequality	TRUE if `$a` is not equal to `$b`.
`$a <> $b`	Inequality	TRUE if `$a` is not equal to `$b`.
`$a !== $b`	Non-identity	TRUE if `$a` is not identical to `$b`.

6.3.8 Operator Precedence

As in many languages, PHP has a set of rules (operator precedence and associativity) that decide how complicated expressions are processed.

Operator precedence are only enforced if you fail to be explicit about your instructions. Unless you have very specific reason to do otherwise, you should always use parentheses in your expressions to make your actual meaning very clear—both to PHP and to others reading your code.

Table 6.8 lists the precedence of operators—with the highest-precedence operators listed at the top of the table. Operators on the same line have equal precedence, in which case their associativity decides which order to evaluate them in.

Table 6.8: Operator Precedence

Associativity	Operators	Type
left	`[`	array()
non-associative	`++ --`	increment/decrement
left	`* / %`	arithmetic
left	`+ - .`	arithmetic and string
left	`<< >>`	bitwise†
non-associative	`< <= > >=`	comparison
non-associative	`== != === !==`	comparison
left	`&`	bitwise† and references†
left	`^`	bitwise†
left	`|`	bitwise†
left	`&&`	logical
left	`||`	logical
left	`? :`	ternary
right	`= += -= *= /= .=`	assignment
	`%= &= |= ^= <<= >>=`	
left	`and`	logical
left	`xor`	logical
left	`or`	logical
left	`,`	many uses

† Not discussed in this Study Book.

6.4 Conditional Statements

Conditional statements are one of the most important features of any language. They allow for the conditional execution of code fragments.

6.4.1 if. . . elsif. . . else

The PHP *if* structure is similar to that of the programming language C.

```
if (expr)
    statement;
```

```
if (expr)
{
    statement;
    ...
    statement;
}
```

The expression "`expr`" is evaluated to its Boolean value. If the expression evaluates to TRUE, PHP will execute the following statement or statement group. If it evaluates to FALSE, PHP will ignore the statement or statement group.

If statements can be nested infinitely within other *if* statements, which provides complete flexibility for conditional execution of blocks of code.

An *if* statement can be extended using *else* to execute a statement or statement group in case the expression in the if statement evaluates to FALSE.

```
if (expr)
{
    statement;
    ...
    statement;
} else {
    statement;
    ...
    statement;
}
```

There can only be one *else* statement associated with an *if* statement.

```
if (expr)
{
    statement;
    ...
    statement;
} elseif (expr) {
    statement;
    ...
    statement;
} else {
    statement;
    ...
    statement;
}
```

Like *else*, the *elseif* statement extends an *if* statement to execute a different statement or statement group if the original *if* expression evaluates to FALSE. However, unlike *else*, it will execute the alternative statement or statement group only if the *elseif* conditional expression evaluates to TRUE.

Exercise 6.11: Modify the code of Script 6.9 so that it prints out the correct day of the week—not just `weekday`.

6.4.2 Switch

The *switch* statement is similar to a series of *ifelse* statements on the same expression. The *switch* statement is used to compare the same variable (or expression) with many different values, and execute a different piece of code depending on which value it equals to.

```
switch (expr)
{
  case value1:
          statement;
          ...
          statement;
  case value2:
          statement;
          ...
          statement;
          break;
  default:
          statement;
          ...
          statement;
}
```

The *switch* statement executes statement by statement. No code is executed until a *case* statement is found with a value that matches the value of the switch expression. PHP continues to execute the statements until the end of the switch block, or the first time it sees a *break* statement. If a break statement does not appear at the end of a case's statement list, PHP will go on executing the statements of the following case or cases. This behaviour is the same as the C programming language.

In a *switch* statement, the condition is evaluated only once and the result is compared to each case statement. In an *elseif* statement, the condition is evaluated each time.

A special case is the *default* case. This case matches anything that was not matched by any of the other cases. The way PHP evaluates the *switch* statement implies that the *default* case should be the last case in the statement.

Exercise 6.12: Modify the code of Script 6.10 so that it prints out the correct day of the week—not just `weekday`.

As will as the name of the day also print the day of the month and day of the year.

6.5 Looping

6.5.1 while

The simplest type of loop in PHP is the *while* loop. They are identical to the *while* loops of C.

While loops are used for the repetitive execution of a block of code. Execution continues only so long as a given condition is true.

```
while (expr)
{
   statement;
   ...
   statement;
}
```

The *while* statement tells PHP to execute the statement or statement group repeatedly, as long as the *while* expression evaluates to TRUE. The value of the expression is checked each time at the beginning of the loop, so even if this value changes during the execution of the statements, execution will not stop until the end of the iteration.

It is possible the while expression evaluates to *False* from the very start—this means that no statement within the loop will be evaluated.

The *break* statement can be used to break out of a *while* loop.

The *continue* statement can be used to skip the rest of the statements of the current iteration and jump to the truth expression evaluation for the next iteration.

6.5.2 do. . . while

The main difference between *while* loops and *do-while* loops is when the truth expression is evaluated. In *while* loops the truth expression is evaluated at the start of the loop–in *do-while* loops the truth expression is evaluated at the end of the loop. This means that *do-while* loops will always evaluate the loop's statements at least once—even if the truth expression evaluates to FALSE the first time through.

The syntax of a *do-while* loop is:

```
do
{
   statement;
   ...
   statement;
} while (expr);
```

The *break* statement can be used to break out of a *do-while* loop.

The *continue* statement can be used to skip the rest of the statements of the current iteration and jump to the truth expression evaluation for the next iteration.

6.5.3 for

The PHP *for* loop behave like their C counterparts. The syntax of a for loop is:

```
for (expr1; expr2; expr3)
{
   statement;
   ...
   statement;
}
```

The first expression (*expr1*) is evaluated once at the beginning of the loop.

At the beginning of each iteration, *expr2* is evaluated. If it evaluates to *True* the loop continues and the statement group is executed. If it evaluates to *False*, the execution of the loop stops.

At the end of each iteration, *expr3* is evaluated.

Each of the expressions can be empty or contain multiple expressions separated by commas. In *expr2*, all expressions separated by a comma are evaluated but the result is taken from the last expression. If *expr2* is empty the loop will be run indefinitely (PHP implicitly considers it as *True*, as in C).

6.5.4 foreach

The *foreach* statement works only with arrays and is used to iterate over the elements within an array. It will issue an error if it is used with any other data type.

There are two syntaxes—the second is a minor but useful extension of the first:

```
foreach (array_expression as $value)
{
   statement;
   ...
   statement;
}

foreach (array_expression as $key => $value)
{
   statement;
   ...
   statement;
}
```

The first form loops over the array given by *array_expression*. On each iteration, the value of the current element is assigned to *$value* and the internal array pointer is advanced by one.

The second form does the same thing, with the addition that the current element's key will be assigned to the variable *$key*.

6.6 Functions

Despite the fact that PHP comes with a large selection of functions that perform many tasks—good programming practices dictates that large scripts be broken into smaller parts or functions. By breaking a large script into smaller functions, the script is easier to understand, control, maintain, and debug.

The syntax of a simple function declaration in PHP is:

```
function function_name( )
{
```

```
    statement;
    ...
    statement;
}
```

Functions are defined using the `function` keyword followed by the function name. Function names follow the naming conventions of variables.

PHP function behave similarly to functions in other programming languages:

- Any valid PHP code may appear inside a function.
- Functions need not be defined before they are referenced (except when a function is defined within a conditional statement—see the PHP manual).
- The function can return a value using the *return* statement. Only one value can be returned. The return value can be of any type—including an array or object.
- Functions can be called recursively.

Script 6.15 shows the use of a simple user defined function.

6.6.1 Function Arguments

Information may be passed to functions via the argument list, which is a comma-delimited list of expressions.

By default arguments are passed to functions by value. PHP also supports default values for arguments, and variable length argument lists.

Script 6.15 shows passing values to a function via the argument list.

Default Argument Values

A function definition as well as defining the arguments to the function can also define default values for the arguments. The default values are used if the function is called with fewer than the defined number of arguments.

The default value must be a constant expression not a variable. It can be a string, scalar or array constant.

When using default arguments, all arguments with defaults should be on the right side of any non-default arguments—otherwise the parser will become confused.

Script 6.15 shows how to set default values for arguments.

6.6.2 Variable Scope

Each variable has a life span in which it exists, this is known as the variables *scope*. It is normal for PHP script to have several variables

called `$var` in existence at one point in time; however, there can only be one active `$var` at any one time.

Global Variables Any variables not set inside a function or an object are considered *global*. That is, they are accessible from anywhere in the script, except inside another function.

Local Variables Variables created or set within a function are local to that function—even if they have the same name as a global variable. Normally variables created within the local scope of a function are unset when program execution leaves the function.

Global Keyword The keyword "global" can be used in functions to declare variables to be "global". That is, all references to variables declared "global" within a function will refer to the global version of the variable. There is no limit to the number of variables that can be declared global within a function.

The variable must exist outside of the function before it can be declared "global" within a function.

Static Keyword A static variable can only exist within a local function scope, but unlike normal local variables the "static" variable does not lose its value when program execution leaves this scope.

6.7 File Handling

There are a number of ways to open and read files in PHP. Each method has its own advantages and disadvantages. The extent of the number of methods in PHP is an indication of the varied uses file handling has when writing scripts.

From within PHP scripts files can be created, deleted, appended, copied, read and written. A survey of all the file handling functions is beyond the scope of this chapter—for a complete list please browse the PHP manual.

Below are some of the the file Input/Output functions available in PHP with a brief description of their usage. For a more complete list of file functions see Section 6.9.3 below and the PHP manual.

Note File handling is Operating System dependent. The most obvious difference is the path separator. Under Windows the path separator is a backslash—which will need to be escaped with another backslash when defining paths to files.

6.7.1 C-style file handling

Many of the file handling functions available in C are also available in PHP. See the PHP manual for the complete list.

fopen() and fclose()

The functions `fopen` opens a file for reading or writing. The function takes two parameters—the file to open, and how it is to be accessed. The first parameter is a string containing the name of the file to open. The second parameter is a string containing a letter defining how the file defined in the first parameter is to accessed—either written to (`w`), appended to (`a`), or read from (`r`).

The `fopen()` function returns a file handle resource that points to a structure in memory that contains information about the opened file. The file handle is passed to reading/writing functions so they know which file is to be read from or written to.

The following is an example of using `fopen()`:

```
$fh_logfile = fopen("${script}.log", "w")
              OR die ("Log file is not writeable!\n");
```

The variable `$fh_logfile` contains the file handle that has to be passed to file access functions.

fread() and fwrite()

The `fread()` function takes two parameters: a file handle to read from (this is the return value from `fopen()`) and the number of bytes to read. Normally the length of a file is unknown—but the end of a file can be tested for using `feof()`, which returns true if you are at the end of the file or false otherwise.

The opposite of `fread()` is `fwrite()`, which also works with the file handle returned by `fopen()`. This function takes a string to write as a second parameter, and an optional third parameter where you can specify how many bytes to write. If you do not specify the third parameter, all of the second parameter is written out to the file.

Script 6.18 shows the C-style file handling functions. Most C-style functions are available in PHP—see Section 6.9.3 or the PHP manual.

6.7.2 High-level file handling

readfile()

The function *readfile* will read a file and output it directly. No form of processing is attempted on the read file. When passed a filename as its only parameter, *readfile()* will attempt to open it, read it all into memory, then output.

If successful, *readfile()* will return an integer equal to the number of bytes read from the file. If unsuccessful, *readfile()* will return FALSE. There are many reasons for failure to read the file—for example, it might not exist, or the web server does not have permission to read the file.

file_get_contents() & file_put_contents()

The function `file_get_contents()` takes one parameter—the name of the file to open. Unlike `readfile()` however, it does not output any data. Instead, it will return the contents of the file as a string, complete with new line characters where appropriate. On failure it will return FALSE.

`file_get_contents()` is equivalent to `fopen`, `fread()` and `fclose()` in one function.

The function `file_put_contents()` is equivalent to `fopen`, `fwrite()` and `fclose()` in one function. It takes two parameters: the filename to write to and the content to write, respectively. If `file_put_contents()` is successful, it will return the number of bytes written to the file; otherwise, it will return FALSE.

You can pass an optional third parameter to `file_put_contents()` which, if set to `FILE_APPEND`, will append the text in your second parameter to the existing text in the file. If you do not use `FILE_APPEND`, the existing text will be overwritten. There are other parameters—see the PHP manual.

file()

The function `file()` takes one parameter—the name of the file to open. It behaves similarly to `file_get_contents()` in that it reads the entire file but instead of returning a string the function will return an array—one line per array element.

6.8 Debugging Scripts

If you are experiencing a problem with your script, the time-honoured way to figure out what's going on is to sprinkle your code with lots of `print` statements.

This method has benefits: it is easy to use, and will generally find the problem through trial and error. The down sides are clear: you need to edit your script quite heavily to make use of the print statements, then you need to re-edit it once you have found the problem to take the print statements back out.

By combining the `print` statement with `var_dump()` to inspect variable contents at various points in the script can speed up error detection.

PHP also has sophisticated error handling functions that can enhance and replace the default error reporting of PHP. These functions documented in the PHP manual but are beyond the scope of this course. See Table 6.14 for a list of some of the error and debugging functions available.

Script 6.7: Using arrays in PHP

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>Using Arrays in PHP</title>
</head>
<body>
<h2>Fruit &amp; Veg</h2>
<?php
     $fruit = array("Bananas", "Apples", "Oranges", "Pears");

     echo "fruit[0]=" . $fruit[0] . "<br>\n";
     echo "fruit[1]=" . $fruit[1] . "<br>\n";
     echo "fruit[3]=" . $fruit[3] . "<br>\n";

     $fruit[4] = "Nectarine";

     echo "fruit[4]=" . $fruit[4] . "<br>\n";


     $veg = array("a"=>"Artichoke", "b"=>"Bean", "c"=>"Carrot");

     echo 'veg["a"]=' . $veg["a"] . "<br>\n";
     echo 'veg["b"]=' . $veg["b"] . "<br>\n";
     echo 'veg["c"]=' . $veg["c"] . "<br>\n";

     $veg["c"] = "Potato";
     $veg["p"] = "Pumpkin";

     echo 'veg["c"]=' . $veg["c"] . "<br>\n";
     echo 'veg["p"]=' . $veg["p"] . "<br>\n";

?>

</body>
</html>
```

Script 6.8: Example of using logical operators and the ternary operator.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>Logical expressions in PHP</title>
</head>
<body>
<h2>Logical expressions</h2>
An example of a logical operator's ability
to short-circuit evaluation.
<p>
<?php
$a = (false &&  print("This will never be printed"));
$b = (true  ||  print("This will never be printed"));
$c = (false and print("This will never be printed"));
$d = (true  or  print("This will never be printed"));

print "<pre>\n";
print "\$a='$a'\n";
$a ? print "\$a is true\n" : print "\$a is false\n";
print "\$b='$b'\n";
print "\$c='$c'\n";
print "\$d='$d'\n";
$b && $d ? print "\$b & \$d are true\n"
         : print "\$b or \$d is false\n";
print "</pre>\n";

?>
</p>
</body>
</html>
```

Script 6.9: Examples of using the *if* statement

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>If statement in PHP</title>
</head>
<body>
<h2>If statement</h2>
<?php
$jd = unixtojd();
$day = JDDayOfWeek($jd);

if ($day == 0)
{
  print "It is Sunday<br>\n";
}
elseif ($day == 6)
{
  print "It is Saturday<br>\n";
}
else
{
  print "It is a Weekday<br>\n";
}
?>
</p>
</body>
</html>
```

Script 6.10: Examples of using the *switch* statement

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>Switch statement in PHP</title>
</head>
<body>
<h2>Switch statement</h2>
<?php
$jd = unixtojd();

switch (JDDayOfWeek($jd))
{
   case 0:
          print "It is Sunday<br>\n";
          break;
   case 1:
   case 2:
   case 3:
   case 4:
   case 5:
          print "It is a Weekday<br>\n";
          break;
   case 6:
          print "It is Saturday<br>\n";
          break;
   default:
          print "Error has occured!<br>\n";
}
?>
</p>
</body>
</html>
```

Script 6.11: Example of using the *while* statement

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>While Statement  in PHP</title>
</head>
<body>
<h2>While Statement</h2>
<?php

$count = 9;

echo "<strong>10...</strong><br>\n";

while ( $count > 0 )
{
  echo "<strong> {$count}...</strong><br>\n";
  $count--;
}

echo "<strong> 0...The End!</strong><br>\n";
?>
</body>
</html>
```

Script 6.12: Example of using the *do-while* statement

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>Do-While Statement  in PHP</title>
</head>
<body>
<h2>Do-While Statement</h2>
<?php

$count = 10;

do {
  if( $count == 10 )
  {
    echo "<strong>10...</strong><br>\n";
  }
  elseif ($count == 0 )
  {
    echo "<strong> 0...The End!</strong><br>\n";
  }
  else
  {
    echo "<strong> {$count}...</strong><br>\n";
  }
  $count--;
} while ( $count >= 0 )
?>
</body>
</html>
```

Script 6.13: Example of using the *for* statement

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>For Statement  in PHP</title>
</head>
<body>
<h2>For Statement</h2>
<?php

for( $count=10; $count>=0; $count--)
{
   if( $count == 10 )
  {
    echo "<strong>10...</strong><br>\n";
  }
  elseif ($count == 0 )
  {
    echo "<strong> 0...The End!</strong><br>\n";
  }
  else
  {
    echo "<strong> {$count}...</strong><br>\n";
  }
}
?>
</body>
</html>
```

Script 6.14: Example of using the *foreach* statement

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>Foreach Statement  in PHP</title>
 <style type="text/css">
   td, th  {text-align: center;
            padding-left: 1em;
            padding-right: 1em;}
 </style>
</head>
<body>
<h2>Foreach Statement</h2>
<table>
<tr>
   <th>Student</th>
   <th>Assignment 1</th>
   <th>Assignment 2</th>
   <th>Assignment 3</th>
</tr>
<?php

$results = array( "W1234567"=>array(78,45,89),
                  "W1236984"=>array(63,33,65),
                  "Q1234567"=>array(56,62,90),
                  "D1234567"=>array(23,43,54));

foreach ( $results as $suid => $marks )
{
 print<<<EOT
 <tr>
 <th>{$suid}</th>
 <td>{$marks[0]}</td>
 <td>{$marks[1]}</td>
 <td>{$marks[2]}</td>
 </tr>
EOT;
}
?>
</table>
</body>
</html>
```

Script 6.15: Example of defining and using a function

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>Functions in PHP</title>
</head>
<body>
<h2>Functions in PHP</h2>
<?php
function reverse($fullname, $comma=false) {
   if ( ! $fullname ) return null;
   $words=explode(" ",$fullname);
   if( count($words)<=1 ) return $fullname;
   $last=array_pop($words);
   if( $comma ) {
      array_unshift($words, $last, ",");
   } else {
      array_unshift($words, $last);
   }
   return implode(" ", $words);
}

echo reverse("John M. Smith"), "<br>\n";
echo reverse("George Jones"), "<br>\n";
echo reverse("Emmanual Kant",true), "<br>\n";
echo reverse("Ren&eacute; Descartes", true), "<br>\n";

?>
</body>
</html>
```

Script 6.16: Example of variable scope

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>Variable Scope in PHP</title>
</head>
<body>
<h2>Variable Scope</h2>
<?php
   function flocal() {
     $name="George F. Jones";
     echo "Function flocal: name=$name<br>\n";
   }
   function fglobal() {
     global $name;
     $name="Samual A. Johnson";
     echo "Function fglobal: name=$name<br>\n";
   }
   $name = "John M. Smith";
   echo "Main script: name=$name<br>\n";
   flocal();
   echo "Main script: name=$name<br>\n";
   fglobal();
   echo "Main script: name=$name<br>\n";
?>
</body>
</html>
```

Script 6.17: Example of static variables and recursion

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>Static variables and Recursion  in PHP</title>
</head>
<body>
<h2>Static Variables and Recursion</h2>
<?php
function countdown()
{
    static $count = 10;

    echo "<strong>{$count}...</strong><br>\n";
    $count--;
    if ($count > 0) {
        countdown();
    } else {
        echo "<strong>0...The End!</strong><br>\n";
    }
}

countdown();
?>
</body>
</html>
```

Script 6.18: Example of using C-style file handling

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
 <title>C-style file handling in PHP</title>
 <style type="text/css">
   .error {color: red;}
 </style>
</head>
<body>
<h2>Reading from and writing to files</h2>
<?php
$basename="script19";
$logfile=$basename . ".log";
$body=$basename . ".php";
$fhlog=fopen($logfile,"a") or
  die("Script19: failed to open logfile");
$fhbody=fopen($body,"r");
if( ! $fhbody ) {
  fwrite($fhlog,"Failed to open file: $body\n");
  fclose($fhlog);
  print '<h1 class="error">';
  print "Failed to open file: $body";
  print '</h1>\n';
 } else {
  print '<pre>';
  while (!feof($fhbody)) {
      $block =  fread($fhbody, 1024);
      print htmlspecialchars($block);
  }
  fclose($fhbody);
  print '</pre>';
  fwrite($fhlog,"Successfully built page\n");
  fclose($fhlog);
 }
?>
</body>
</html>
```

6.9 Builtin Functions

6.9.1 String Functions

Table 6.9: List of most, but not all of the PHP built-in string functions

Function	Description
addslashes	Quote string with slashes
chr	Return a specific character
chunk_split	Split a string into smaller chunks
count_chars	Return information about characters used in a string
crypt	One-way string encryption (hashing)
echo	Output one or more strings
explode	Split a string by string
fprintf	Write a formatted string to a stream
html_entity_decode	Convert all HTML entities to their applicable characters
htmlentities	Convert all applicable characters to HTML entities
htmlspecialchars_decode	Convert special HTML entities back to characters
htmlspecialchars	Convert special characters to HTML entities
implode	Join array elements with a string
join	Alias of implode()
ltrim	Strip whitespace (or other characters) from the beginning of a string
nl2br	Inserts HTML line breaks before all newlines in a string
ord	Return ASCII value of character
parse_str	Parses the URL query string into variables
print	Output a string
printf	Output a formatted string
quotemeta	Quote meta characters
rtrim	Strip whitespace (or other characters) from the end of a string
sprintf	Return a formatted string
sscanf	Parses input from a string according to a format
str_ireplace	Case-insensitive version of str_replace().
str_pad	Pad a string to a certain length with another string
str_repeat	Repeat a string
str_replace	Replace all occurrences of the search string with the replacement string
str_split	Convert a string to an array
str_word_count	Return information about words used in a string
strcasecmp	Binary safe case-insensitive string comparison
strchr	Alias of strstr().
strcmp	Binary safe string comparison
strcoll	Locale based string comparison
strcspn	Find length of initial segment not matching mask
strip_tags	Strip HTML and PHP tags from a string
stripos	Find position of first occurrence of a case-insensitive string
stripslashes	Un-quote string quoted with addslashes().

Continued on next page

Continued from previous page

Function	Description
stristr	Case-insensitive strstr().
strlen	Get string length
strncasecmp	Binary safe case-insensitive string comparison of the first n characters
strncmp	Binary safe string comparison of the first n characters
strpbrk	Search a string for any of a set of characters
strpos	Find position of first occurrence of a string
strrchr	Find the last occurrence of a character in a string
strrev	Reverse a string
strripos	Find position of last occurrence of a case-insensitive string in a string
strrpos	Find position of last occurrence of a char in a string
strspn	Find length of initial segment matching mask
strstr	Find first occurrence of a string
strtok	Tokenize string
strtolower	Make a string lowercase
strtoupper	Make a string uppercase
strtr	Translate certain characters
substr_compare	Binary safe optionally case insensitive comparison of 2 strings from an offset, up to length characters
substr_count	Count the number of substring occurrences
substr_replace	Replace text within a portion of a string
substr	Return part of a string
trim	Strip whitespace (or other characters) from the beginning and end of a string
ucfirst	Make a string's first character uppercase
ucwords	Uppercase the first character of each word in a string
vfprintf	Write a formatted string to a stream
vprintf	Output a formatted string
vsprintf	Return a formatted string
wordwrap	Wraps a string to a given number of characters using a string break character

6.9.2 Array Functions

Table 6.10: List of most, but not all of the PHP built-in array functions

Function	Description
array_combine	Creates an array by using one array for keys and another for its values
array_count_values	Counts all the values of an array
array_fill	Fill an array with values
array_filter	Filters elements of an array using a callback function
array_flip	Exchanges all keys with their associated values in an array
array_key_exists	Checks if the given key or index exists in the array
array_keys	Return all the keys of an array
array_map	Applies the callback to the elements of the given arrays

Continued on next page

Continued from previous page

Function	Description
array_merge	Merge one or more arrays
array_multisort	Sort multiple or multi-dimensional arrays
array_pad	Pad array to the specified length with a value
array_pop	Pop the element off the end of array
array_product	Calculate the product of values in an array
array_push	Push one or more elements onto the end of array
array_rand	Pick one or more random entries out of an array
array_reduce	Iteratively reduce the array to a single value using a callback function
array_reverse	Return an array with elements in reverse order
array_search	Searches the array for a given value and returns the corresponding key if successful
array_shift	Shift an element off the beginning of array
array_slice	Extract a slice of the array
array_splice	Remove a portion of the array and replace it with something else
array_sum	Calculate the sum of values in an array
array_unique	Removes duplicate values from an array
array_unshift	Prepend one or more elements to the beginning of an array
array_values	Return all the values of an array
array_walk_recursive	Apply a user function recursively to every member of an array
array_walk	Apply a user function to every member of an array
array	Create an array
arsort	Sort an array in reverse order and maintain index association
asort	Sort an array and maintain index association
compact	Create array containing variables and their values
count	Count elements in an array, or properties in an object
current	Return the current element in an array
each	Return the current key and value pair from an array and advance the array cursor
end	Set the internal pointer of an array to its last element
extract	Import variables into the current symbol table from an array
in_array	Checks if a value exists in an array
key	Fetch a key from an associative array
krsort	Sort an array by key in reverse order
ksort	Sort an array by key
list	Assign variables as if they were an array
next	Advance the internal array pointer of an array
prev	Rewind the internal array pointer
range	Create an array containing a range of elements
reset	Set the internal pointer of an array to its first element
rsort	Sort an array in reverse order
shuffle	Shuffle an array
sort	Sort an array
uasort	Sort an array with a user-defined comparison function and maintain index association

Continued on next page

Continued from previous page

Function	Description
uksort	Sort an array by keys using a user-defined comparison function
usort	Sort an array by values using a user-defined comparison function

6.9.3 File Functions

Table 6.11: List of most, but not all of the PHP built-in file functions

Function	Description
basename	Returns filename component of path
copy	Copies file
delete	See unlink or unset
dirname	Returns directory name component of path
fclose	Closes an open file pointer
feof	Tests for end-of-file on a file pointer
fflush	Flushes the output to a file
fgetc	Gets character from file pointer
fgetcsv	Gets line from file pointer and parse for CSV fields
fgets	Gets line from file pointer
fgetss	Gets line from file pointer and strip HTML tags
file_exists	Checks whether a file or directory exists
file_get_contents	Reads entire file into a string
file_put_contents	Write a string to a file
file	Reads entire file into an array
filesize	Gets file size
filetype	Gets file type
fnmatch	Match filename against a pattern
fopen	Opens file or URL
fpassthru	Output all remaining data on a file pointer
fputcsv	Format line as CSV and write to file pointer
fputs	Alias of fwrite
fread	Binary-safe file read
fscanf	Parses input from a file according to a format
fseek	Seeks on a file pointer
fwrite	Binary-safe file write
glob	Find pathnames matching a pattern
is_dir	Tells whether the filename is a directory
is_executable	Tells whether the filename is executable
is_file	Tells whether the filename is a regular file
is_readable	Tells whether the filename is readable
is_writable	Tells whether the filename is writable
is_writeable	Alias of is_writable
mkdir	Makes directory
pathinfo	Returns information about a file path
pclose	Closes process file pointer

Continued on next page

Continued from previous page

Function	Description
popen	Opens process file pointer
readfile	Outputs a file
realpath	Returns canonicalized absolute pathname
rename	Renames a file or directory
rmdir	Removes directory
tempnam	Create file with unique file name
tmpfile	Creates a temporary file
unlink	Deletes a file

6.9.4 Variable Handling Functions

Table 6.12: List of most, but not all of the PHP built-in variable handling functions

Function	Description
doubleval	Alias of floatval
empty	Determine whether a variable is empty
floatval	Get float value of a variable
gettype	Get the type of a variable
intval	Get the integer value of a variable
is_array	Finds whether a variable is an array
is_binary	Finds whether a variable is a native binary string
is_bool	Finds out whether a variable is a boolean
is_buffer	Finds whether a variable is a native unicode or binary string
is_callable	Verify that the contents of a variable can be called as a function
is_double	Alias of is_float
is_float	Finds whether the type of a variable is float
is_int	Find whether the type of a variable is integer
is_integer	Alias of is_int
is_long	Alias of is_int
is_null	Finds whether a variable is NULL
is_numeric	Finds whether a variable is a number or a numeric string
is_real	Alias of is_float
is_scalar	Finds whether a variable is a scalar
is_string	Find whether the type of a variable is string
is_unicode	Finds whether a variable is a unicode string
isset	Determine whether a variable is set
print_r	Prints human-readable information about a variable
settype	Set the type of a variable
strval	Get string value of a variable
unset	Unset a given variable
var_dump	Dumps information about a variable
var_export	Outputs or returns a parsable string representation of a variable

6.9.5 Perl Regular Expression Functions

Table 6.13: List of most, but not all of the PHP built-in PCRE Regular Expression functions

Function	Description
preg_grep	Return array entries that match the pattern
preg_last_error	Returns the error code of the last PCRE regex execution
preg_match_all	Perform a global regular expression match
preg_match	Perform a regular expression match
preg_quote	Quote regular expression characters
preg_replace	Perform a regular expression search and replace
preg_split	Split string by a regular expression

6.9.6 Error and Debugging Functions

Table 6.14: List of some of the Error functions available in PHP functions

Function	Description
assert_options	Set/get the various assert flags
assert	Checks if assertion is FALSE
debug_backtrace	Generates a backtrace
debug_print_backtrace	Prints a backtrace
error_get_last	Get the last occurred error
error_log	Send an error message somewhere
error_reporting	Sets which PHP errors are reported
restore_error_handler	Restores the previous error handler function
restore_exception_handler	Restores the previously defined exception handler function
set_error_handler	Sets a user-defined error handler function
set_exception_handler	Sets a user-defined exception handler function
trigger_error	Generates a user-level error/warning/notice message

6.10 Questions

Short Answer Questions

Q. 6.13: Explain what the term "to parse" means.

Q. 6.14: How does the PHP module recognise PHP commands embedded in a HTML document?

Q. 6.15: How does the Apache server know that a particular document may have PHP commands embedded in it?

Q. 6.16: What are escape characters in a string?

Q. 6.17: What is a "boolean" test of a variable?

Q. 6.18: How is a boolean test done on variables that are not defined as "boolean"—that is integers, floats arrays and strings?

Q. 6.19: Explain the difference between "strongly typed", "loosely typed" and "typeless" languages. Give examples.

Q. 6.20: What does "the scope" of a variable mean? What is the scope of a variable defined within a PHP function?

Q. 6.21: Explain the difference when a string of characters are enclosed in single quotes and double quotes.

Q. 6.22: What is a "regular expression"?

Q. 6.23: Explain why the equals character "=" is called the "assignment" operator and not "equals".

Q. 6.24: Explain how a string can be "greater" or "less" than another string.

Q. 6.25: Why is it necessary to define operator precedence. Give examples.

Q. 6.26: Give examples of using the `break` and `continue` keywords within a loop. How are they different?

Q. 6.27: The "fopen" function returns a "file handle"—what is a file handle? Why is it needed?

Q. 6.28: When using the "fopen" function how do you know that it has succeeded? Write a code snippet to illustrate your answer.

6.11 Further Reading and References

- PHP home site—www.php.net
- The best reference for PHP is the PHP manual itself. The manual can be found on the course web site.
- The online PHP manual (at the PHP site) is extremely useful as it is dynamic and has user additions to it.

II. Server Side

Chapter 7 HyperText Transfer Protocol

The Hypertext transfer protocol is the language the web client and web server talk to each other. Like many protocols it is a clear text language designed for computer and human interpretation. Understanding the language spoken by web clients and servers is vital if you wish to be able to administer a web site.

The version of the Hypertext transfer protocol we will be describing here is 1.1 This module is not a definitive reference for the transfer protocol, but covers the main aspects of the language. The course resources directory contains a copy of the complete definition of the language, RFC2616.

Chapter contents

When, using your favourite web browser, you connect to a site and the browser starts downloading pages and images, it is holding an on going dialog with a web server running on the remote computer. Irrespective of the type of computer you have connected to, the web server and web browser speak the same language.

When you request a URL, the browser opens a connection to the server indicated in the URL, sends a request for the file, receives a reply, and then displays the contents of the reply for you.

The Hypertext Transfer Protocol (HTTP) is the language that is used to enable the web browser and the web server to talk to each other so that your request ("fetch and display a URL") can be completed.

The short conversation between the browser and the server is conducted using the International Standards Organisation (ISO) Latin-1 alphabet and is human readable, with carriage return/line feed pairs used to separate lines. It normally consists of two phases, the *request phase* from the browser, followed by the *response phase* from the server.

Example 7.1: A simple and informative way to learn about the HTTP is to connect to an HTTP server using the program `telnet` and talk to the server directly. The following assumes you are using a Unix system but any system with a `telnet` application[1] that allows you to specify the port will work.

To fetch the URL

`htttp://www.sci.usq.edu.au/index.html`

type the following at the command prompt

```
telnet www.sci.usq.edu.au 80
```

This command directs telnet to connect to `www.sci.usq.edu.au` using the HTTP port 80.

Then type the following request lines *exactly*, as they are case sensitive. The string `<ret>` represents a return or the `Enter` key.

```
GET /index.html HTTP/1.1<ret>
Host: www.sci.usq.edu.au<ret>
<ret>
```

Do not forget that last return.

The server should send you back the document.

An example of connecting to a web server and requesting a document follows.

```
>> telnet www.sci.usq.edu.au 80
Trying 139.86.138.50...
Connected to www.sci.usq.edu.au.
Escape character is '^]'.
GET /index.html HTTP/1.1
Host: www.sci.usq.edu.au

HTTP/1.1 200 OK
Date: Wed, 04 Feb 2009 05:08:17 GMT
Server: Apache/2.2.8 (Unix)
Last-Modified: Fri, 11 Feb 2005 03:10:10 GMT
ETag: "13-1337-3efcda2df2880"
Accept-Ranges: bytes
Content-Length: 4919
Content-Type: text/html

<html>
<head>
<title>USQ Maths &amp;amp; Computing Home Page</title>
...
(The rest of the document follows)
...
</body>
</html>
```

[1] PuTTy is a telnet and ssh client for Windows

After the requested document is received the connection remains open for further requests. The connection can be terminated by exiting from `telnet`[2]

Exercise 7.2: Connect to your personal web server[3] using `telnet` and request the text file `/welcome.txt`

After connecting via `telnet`, type the commands

```
GET /welcome.txt HTTP/1.1<ret>
Host: localhost<ret>
<ret>
```

What is the response from your web server?

Exercise 7.3: Repeat Example 7.1, but this time use the request

```
GET /index.html HTTP/1.0<ret>
<ret>
```

How is this response different?[4]

The conversation between client and server is summarised in the Table 7.1.

7.1 Request Phase

The request phase consists of the request line followed by an optional request header. The request line consists of the request method, the path part of the URL requested, and the HTTP version number. The request line is *case sensitive*.

If you requested the URL `http://www.sci.usq.edu.au/home.html` the request line would be the following (assuming the request method `GET` is being used)

```
GET /home.html HTTP/1.1
```

After the request line comes the request header, containing any number of header fields. These fields are mostly informational, and all are optional, except the `Host` request header field. The `Host` request

[2] The usual escape character is `^]`, that is the `]`-key depressed while the control-key is held down.

[3] If your web server has been installed on your own machine then its address will be the name you gave your machine when you installed the operating system. If you are running Linux then it will be `127.0.0.1` or `localhost.localdomain`. If neither work, then look for the name of your machine in the file `/etc/HOSTNAME`.
If nothing seems to work check that you have started your web server!

[4] Using HTTP/1.1, the server keeps the connection open after it has sent the requested document. The default action in HTTP/1.0 was to close the connection after every request. With the increase in http requests this placed a high overhead on the Internet as every request had to renegotiate the connection. To reduce the connecting overhead the default behaviour was changed. Servers may not conform to this default behaviour and still close the connection after the requested document has been sent. See the request header `Connection` on how to force persistent connections.

Client Request	Request Line	Contains the *Request Method* and the document path that has been requested
	Request Headers	Information for the server that can modify the request.
	Request Data	Depending on the *Method* this could be empty, or could be the contents of a file or data from a forms page
Server Response	Response Line	Contains the *Status Code* for the response.
	Response Headers	Information for the client that can tell it (among other things) the length of the document to follow.
	Response Data	Depending on the *request method* and *response status code* the document would be sent at this stage

Table 7.1: Summary of the conversation between the web client and the web server.

header field *must* accompany all HTTP/1.1 requests. This field specifies the Internet host (and optionally the port number) of the server containing the resource requested[5].

Note The request header must be terminated with a *blank line*

7.1.1 The Request Method

The *request method* specifies what action is required of the server. The most commonly used request methods are `GET`, `POST`, `HEAD`, and `PUT`. The request line is *case sensitive* so the request methods must be specified in uppercase.

Common request methods (This list is not complete, see RFC2616, in the course resources directory, for the complete list of methods):

`GET` Retrieve the resource as identified by the requested URL. If the requested URL refers to a server script, then the server returns the data produced by the script. If the request URL refers to a file then the server returns the file.

`HEAD` Only retrieve the header information of the resource identified by the requested URL. The server will not send the resource itself.

`POST` One of the main functions of the POST method is to send data entered on a form page (see Module 9) to a process controlled by the server. The server will return the output from the process.

[5] The `Host` field was introduced to allow virtual sites to be created. When the server accepts a connection from a web client it does not know the name of the machine the client thinks it has connected to. This meant that only one web site could be maintained on any machine. By including the `Host` header name the server now knows which site you wanted. This means that any number of web sites can be maintained on one machine by one server.

`PUT` The PUT method allows the browser to request that the enclosed data be stored on the server under the supplied request URL. This method is used to publish resources on a server.

`DELETE` This method requests that the server delete the resource identified by the request URL.

Exercise 7.4: Connect to your own server and use the `GET` method to request the file `/welcome.txt`.

Now use the `HEAD` method. For example

```
HEAD /welcome.txt HTTP/1.1<cr>
Host: your_server_machine_name:port<cr>
<cr>
```

What is the difference between the two methods? What could be the purpose of the `HEAD` method?

7.1.2 The Request Header

Following the request-line are the optional request header fields. These fields allow the client to pass additional information to the server. The additional information can be used as request modifiers. The request headers have varying uses and importance depending on the request method used.

Below are some of the request header fields that can be specified by the browser. This list is not necessarily complete and for a fuller explanation (thought not necessarily more informative) of the meaning of each header see RFC2616, in the course resources directory.

`Accept` The client can use this header to specify a list of acceptable media types. Multiple lines of this type are allowed. The `Accept` field is specified as a MIME type, and can use an asterisk as a wild card.

For example, if a browser is willing to accept plain text documents, html documents, and only GIF and JPEG images the request header would contain the following lines

```
Accept:  text/plain
Accept:  text/html
Accept:  image/gif
Accept:  image/jpg
```

The browser is also allowed to qualify its acceptance with a quality value. The `q` value ranges from 0.0, meaning "least preferred" to 1.0 meaning "most preferred". So a request header telling the server that the audio format most preferred is Microsoft's `wav` format, somewhat preferred is the AIFF sound format and all other formats are least preferred

```
Accept:  audio/x-aiff; q=0.5
Accept:  audio/x-wav; q=1.0
```

```
Accept:  audio/*; q=0.1
```

If the server has the same sound document in multiple formats it can send the preferred audio format of the requesting browser.

If no Accept header field is present, then it is assumed that the client accepts all media types. If an Accept header field is present, and if the server cannot send a response which is acceptable, then the server should send a 406 (not acceptable) status code (see §7.2.1).

`Accept-Charset` This header can be used to specify the list of character sets acceptable to the client. The ISO 8859-1 (Latin-1) character set is assumed to be acceptable to all clients. Quality values can be employed to list preferred character sets.

`Accept-Encoding` Restricts the content encoding acceptable to the client. That is the compression method(s) that the client will accept.

This field allows resources to be compressed without losing the underlying mime type.

`Accept-Language` Language(s) acceptable to the client.

`Authorization` Used in various authorisation/verification schemes. A browser that wishes to authenticate itself with a server (usually, but not necessarily, after receiving a 401 status code, see Table 7.4) may do so by including an Authorization request-header field with the request.

See Module 12 on authorisation.

`Content-length` This field specifies the length, in bytes, of the data that will follow the request headers. This is used with `POST` and `PUT` methods. This field is mandatory with both these methods.

`Cookie` This field contains *all* the matching *magic cookies* that the browser has stored. See the Module 10 on how to use cookies.

Strictly speaking this is not part of the HTTP/1.1 protocol but originally a Netscape Communications Corp. extension. It has since been published as RFC2109 (see the course resource directory).

`Host` This field specifies the Internet host and port number of the resource being requested. It is obtained from the URL given by the user. A client must include a Host header field in all HTTP/1.1 request messages. This is the only mandatory field. If it is missing the server should respond with a 400 *Bad request* status code (see Table 7.4).

`If-Modified-Since` Return the resource only if it has been modified since the date specified. If the document has not been modified the server returns a 304 *Not Modified* status code (see Table 7.3).

For efficiency browsers cache documents locally. If a local copy exists the local copy of a document will be displayed, but the server needs to be queried to see if the document has changed. If

it has changed then a new copy must be obtained from the server. This field can be used to request modified documents only.

All times and dates in header fields are specified in Greenwich Mean Time (GMT), also known as Universal Time (UT).

`Range` If the client has a partial copy of the resource, the Range field can be used to retrieve the missing part. For example

```
Range:  bytes=100-500
```

If the Range field is used with a conditional `GET` (If-Modified-Since *etc.*) and the document has been modified the `GET` fails.

`If-Range` If the client has a partial copy of the resource, this field can be used to retrieve the missing part. If the resource has been modified since the date specified the entire resource will be sent.

The range is specified in bytes.

```
If-Range:  bytes=100-500, Sat, 29 Jan 2000
```

All times and dates in header fields are specified in Greenwich Mean Time (GMT).

`If-Unmodified-Since` Return the resource only if it has **not** been modified since the date specified.

All times and dates in header fields are specified in Greenwich Mean Time (GMT).

Exercise 7.5: Using `telnet` request a document from your server and use the header field `Range` to specify a range in bytes.

For example

```
GET /index.html HTTP/1.1<ret>
Host: your_server_machine_name:port<ret>
Range: bytes=5-105<ret>
<ret>
```

What is the response of the server? Compare the servers response without the `Range` header. Apart from the partially sent resource what else is different in the server's response?

The request header fields fall into two categories informational or modifying. The informational fields send information to the server that it can choose to act on or not. The modifying fields modify the response of the server to the request method.

The HTTP/1.1 specifies only one field as mandatory in the request headers, that is the `Host` field. The host field contains the machine name (and optional port number) contained in the request URL. This field allows the use of virtual hosts. Virtual hosts allow a machine with only one IP address to have multiple domain names and therefore can appear as different web sites. An example of virtual hosts are the two web sites `www.sci.usq.edu.au` and `jamsb.austms.org.au`. The sites have completely different domain names but the web server, the machine and IP address are the same. The host field allows the server

to identify which site you wish to communicate with and therefore which document tree you can access.

7.1.3 The Request Data

After the *request header* and a *blank line*, the client has the option of sending data, if it has made a `POST` or `PUT` request. There is no restriction on the format or type of data, only that it be `Content-length` bytes long.

If the client sent a `GET`, `HEAD`, or `DELETE` request, there is nothing more to send. At this point the client has finished the request and waits for the response from the server.

7.2 Response Phase

After the request has been sent by the client it is the response phase of the server. It sends back to the client the response-line followed by the optional response headers, ultimately followed by the data, if any, requested.

The response-line consists of the HTTP version, a three digit numerical status code, and a text explanation of the status code.

7.2.1 Response Status Codes

The status code is a 3 digit integer that the server sends. It specifies the result of the attempt to understand and satisfy the browsers request. A reason phrase is defined for each status code and returned with the code. The status code is intended for machine interpretation while the reason phrase is intended for humans. Though in some (if not all) instances the *reason phrase* is as opaque as the *status code*!

The first digit of the status code defines the type of response. The last two digits do not have any categorisation role. There are 5 values for the first digit:

1xx Informational	Request received, and continuing to process.
2xx Success	The action was successfully received, understood, and accepted.
3xx Redirection	Further action on the part of the browser must be taken in order to complete the request.
4xx Client Error	The request contains bad syntax or cannot be fulfilled.
5xx Server Error	The server failed to fulfil an apparently valid request.

Success Codes 2xx

This class of status code indicates that the client's request was successfully received, understood, and accepted.

Exercise 7.6: Using `telnet` repeat exercise 7.2. Identify the response header from the server.

Code	Text	Meaning
200	OK	The request has succeeded.
201	Created	The request has been fulfilled and resulted in a new resource being created. The result of a successful `PUT` request.
202	Accepted	The request has been accepted for processing, but the processing has not been completed.
203	Non-Authoritative Information	The information returned in the header is not the definitive or authoritative set available.
204	No Content	The server has fulfilled the request but there is no new information to send back. The web browser should not change the current document view.
205	Reset Content	The server has fulfilled the request and the web client should reset the document. This response is primarily intended to allow a form to be cleared so that the user can easily initiate another input action.
206	Partial Content	The server has fulfilled the partial GET request for the resource. The initiating request must have included a Range header field indicating the desired range.

Table 7.2: Success codes 2xx returned by the server

Exercise 7.7: Using telnet request a document from your server. One of the header fields returned by the server with the document is the `Last-Modified` date. The date the document was last modified. This date, like all HTTP dates is in Greenwich Mean Time (GMT).

Now request the document and add the `If-Unmodified-Since` field to your request.

What is the servers response when the `If-Unmodified-Since` date is the same as the `Last-Modified` date of the document? If it is earlier? If it is later?

What are the server codes returned?

(The Linux commands `date -u` will return the current date in Universal Time, UT which is the same as GMT)

Redirection Codes 3xx

This class of status codes indicate that further action needs to be taken by the browser in order to fulfil the request. That is the URLs exist but have moved. The server uses the *Location* header field to specify where the document can now be found.

Table 7.3 lists some of the redirection codes 3xx, returned by the server

Code	Text	Meaning
300	Multiple Choices	The requested resource corresponds to any one of a set of representations, each with its own specific location. Information is provided so that the user (or web client automatically) can select a preferred representation and redirect its request to that location.
301	Moved Permanently	The requested resource has been assigned a new permanent URL and any future references to this resource should be done using one of the returned URLs.
302	Moved Temporarily	The requested resource can be found temporarily at a different URL.
303	See Other	The response to the request can be found under a different URL and should be retrieved using a GET method. This code exists primarily to allow the output of a POST activated script to redirect the web client to a selected resource.
304	Not Modified	If the client has performed a conditional GET request and access is allowed, but the document has not been modified, the server responds with this status code.

Table 7.3: Some of the redirection codes 3xx, returned by the server

Exercise 7.8:

Using a browser request the *directory*

`http://www.sci.usq.edu.au/courses`

Now using `telnet` request the directory again using the command

```
GET /courses HTTP/1.1<ret>
Host: www.sci.usq.edu.au<ret>
<ret>
```

What is the server response code? Explain the response from the server.

Now request the directory, with

```
GET /courses/ HTTP/1.1<ret>
Host: www.sci.usq.edu.au<ret>
<ret>
```

Note: The requested resource is now `/courses/`.

Explain the steps the browser must have gone through to download the document.

Explain why when accessing directories the complete directory name `courses/` should always be used.

Client Error Codes 4xx

This class of error codes are intended for the cases when the web client seems to have made an error. A client error can be due to syntax error so that the request could not be understood by the server (Status 400), or due to an authorisation error (Status 401). That is, to retrieve the document the client needs to include authentication headers.

Code	Text	Meaning
400	Bad Request	The request could not be understood by the server.
401	Unauthorized	The request requires user authentication.
403	Forbidden	The server understood the request, but is refusing to fulfil it.
404	Not Found	The server has not found anything matching the requested URL.
405	Method Not Allowed	The method specified in the request line is not allowed for the requested resource.
406	Not Acceptable	The data to be returned to the client would be unacceptable. This decision by the server is based on the accept headers sent by the client.
410	Gone	The requested resource is no longer available at the server and no forwarding address is known.
411	Length Required	The server refuses to accept the request without a defined `Content-Length`. This would be sent if a `POST` method was used in the request and no *Content-Length* was supplied in the header fields.
412	Precondition Failed	The precondition given in one or more of the request-header fields evaluated to false when it was tested on the server.

Table 7.4: Some of the client error codes 4xx returned by the Server

Table 7.4 lists some of the client error codes 4xx, returned by the server

Exercise 7.9: Using `telnet` request a document from the server, without sending the `Host` field. For example,

```
GET /index.html HTTP/1.1<ret>
<ret>
```

What was the server's response.

Exercise 7.10: Using a browser download the document

`http://www.sci.usq.edu.au/dept/`

Note the response from the browser/server.

Now download the same document using `telnet`.

What was the response of the browser when it recognised the server status code?

Exercise 7.11: Using `telnet` download a non-existent document from your server.

What is the response?

Download the same non-existent document using a web browser.

Server Error Codes 5xx

These error codes are intended for the cases in which the server is aware that it has erred or is incapable of performing the request. Table 7.5 lists some of the server error codes 5xx, returned by the server

Code	Text	Meaning
500	Internal Server Error	The server encountered an unexpected condition which prevented it from fulfilling the request.
501	Not Implemented	The server does not support the functionality required to fulfil the request. This is the appropriate response when the server does not recognise the request method and is not capable of supporting it for any resource.
503	Service Unavailable	The server is currently unable to handle the request. This condition is temporary. Possible causes for the condition are server overload or server maintenance
505	HTTP Version Not Supported	The server does not support, or refuses to support, the HTTP protocol version that was used in the request message.

Table 7.5: Some of the server error codes 5xx, returned by the Server

Exercise 7.12: Using `telnet` request a document from your server using an unknown method. For example

```
XXX /index.html HTTP/1.0<ret>
```

What is the server's response.

Exercise 7.13: Using `telnet` request a document from your server using an unknown HTTP version. For example

```
XXX /index.html HTTP/2.0<ret>
Host: your-server-name<ret>
<ret>
```

What is the server's response. Is the server's response the correct response?

7.2.2 The Response Header

After the status line, the server sends a response header. The header contains information that applies to the server itself and information that applies to the document to follow. Like the *request header*, much of the information in the response header is optional.

Some of the server response headers are:

`Server` The name and version of the server.

`Date` The current date and time (GMT).

`Expires` The date at which the document expires.

`Allow` The requests that the requesting user can issue. For example

```
Allow:  GET, POST
```

`Last-Modified` Date at which the document was last modified (GMT).

`Location` The location of the document in a redirection response.

`Pragma` Used to give hints to the browser or proxy server, such as

```
Pragma:  no-cache
```

That is, don't cache the document locally.

`Content-Length` Length, in bytes, of the data to follow. For example,

```
Content-Length:  12456
```

`Content-Base` Can be used to specify the base URL for resolving relative URLs within the document to follow. If it is absent the URL of the document to follow is used as the base URL.

`Content-Type` MIME media-type of the data to follow. For example,

`Content-Type:  text/html`.

(See Module 8, on MIME typing)

`Content-Range` When the server sends only part of a document this header specifies where in the full document the partial document should be inserted. It also indicates the total size of the full document.

`Content-Encoding` When present, its value indicates what additional content codings have been applied to the document to follow, and thus what decoding mechanisms must be applied by the browser to obtain the MIME media-type referenced by the Content-Type header field.

Content-Encoding is primarily used to allow a document to be compressed without losing the identity of its underlying MIME media type.

`Content-Language` Describes the natural language(s) of the intended audience for the following document.

`WWW-authenticate` This header field must be included in 401 (Unauthorized) response messages. The field value consists of at least one challenge that indicates the authentication or validation scheme(s)

(and parameters) that need to be used to get access to the requested document.

`Set-Cookie` Header field used to pass a *Cookie* to the browser. (See §10.7.5 on using cookies).

The most important header field which is *not optional*, is the `Content-Type` field. This field is essential, without it, the browser will not know how to display the document to be sent.

Header fields such as *Server*, *Date*, *Last-Modified*, and *Expires* are informational. The last two can be used by smart browsers or proxies to cache documents locally to be reused without fetching them over the network. All dates used by HTTP are in Greenwich Mean Time (GMT, also known as UT, Universal Time and UST, Universal Standard Time).

The `Location` header is used in conjunction with the redirection messages 301 and 302. Both of these status codes tell the browser that the requested document is located elsewhere. When the response line contains a 301 or 302 status code then the `Location` header is added and contains the URL of the new location of the document.

The `Content-Length` gives the size, in bytes, of the document to follow.

Exercise 7.14: Repeat exercises 7.4,7.5, 7.8, 7.10 using `telnet`, and this time study the server header fields and how they change depending on the request.

7.2.3 The Response Data

Last, but not least, the server sends the document itself!

After the last header field the server sends a blank line. If the browser requested just the header information using the `HEAD` request method, the server has fulfilled the request. If the connection is not to be *kept alive* the server closes the connection[6]

Under the the HTTP/1.1 the default behaviour is for the server to keep the connection alive and wait for a new request from the browser. If the request is slow in coming the open connection will time out and be closed by the server.

The HTTP doesn't require special treatment for different data types, it only recognises a binary data stream. Neither does the protocol put a limit on the size of the documents sent. Anything from the 12-byte message *Hello World!* to multi-megabyte files are equally acceptable.

7.3 Questions

Short Answer Questions

Q. 7.15: What are the two phases of an HTTP request?

[6] The browser initiates the connection but it is the server that decides to close the connection!

Q. 7.16: What is one of the differences between HTTP/1.1 and HTTP/1.0?

Q. 7.17: What are the 3 parts of the *Response Phase*?

Q. 7.18: What are the 3 parts of the *Request Phase*?

Q. 7.19: What is the information sent in the *request line*?

Q. 7.20: Describe the request method GET.

Q. 7.21: What is the purpose of the HEAD method?

Q. 7.22: What is the purpose of the *request header*?

Q. 7.23: HTTP/1.1 has one mandatory header field, what is it? Why is it a mandatory header?

Q. 7.24: What is the response from the server to a `HEAD` request?

Q. 7.25: Why would a client send a `HEAD` request?

Q. 7.26: What is the *status code*? Who sends it? Why is it sent?

Q. 7.27: What is the meaning of the first digit in a status code?

Q. 7.28: How are redirection codes interpreted by the client?

Q. 7.29: What is the response header?

Q. 7.30: Why is the `Content-Type` header so important?

Q. 7.31: In what format is the actual document data sent by the server?

7.4 Further Reading and References

- The obvious place to start for more information on the Hypertext Transfer Protocol is the *Request For Comment* document that defines the HTTP/1.1. The document is RFC2616 from the Network Working Group. A copy of RFC2616 can be found in the course resources directory.
- As always the main site for all information on current and future web standards is the World Wide Web Consortium at

  http://www.w3c.org/

Chapter 8 Multipurpose Internet Mail Extensions

Multipurpose Internet Mail Extensions or MIME was developed to allow multimedia files to be sent using electronic mail. This module will show how the existing MIME system was incorporated into the web.

Chapter contents

8.1 MIME types

Historically electronic mail was developed to send text messages using the Internet. The protocol (SMTP: Simple Mail Transfer Protocol) was designed to accept only the 96 printable characters on an English/US keyboard. This made it impossible to send binary files via email unless they where encoded. A number of encoding schemes have been developed that allow binary files to be transferred, for Unix there is uuencode/uudecode, for the Macintosh there is binhex. All these schemes have one thing in common, they encode the bytes of a binary file to produce a file that contains only the (at most) 96 printable characters of the ASCII character set.

8.1.1 Base64 Encoding

The encoding that has become standard with electronic mail is Base64. This scheme encodes any binary file into a file that contains only 65 characters (64 characters and the padding character "="). The characters (in order) are

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
0123456789+/
```
= (padding)

any character found in a Base64 encoded file that is not one of the 65 characters above are ignored.

To convert any file into a file containing only the 64 characters above, Base64 converts each 3 bytes of the input stream (reading left to right) into 4 output bytes. To do this the encoder splits the 24-bit input

group into 4 6-bit groups. Each 6-bit group represents one character from the Base64 alphabet above. Using the characters above and starting at zero the characters are numbered sequentially.

A full encoded 24-bits are always completed at the end of a body. When fewer than 24 input bits are available in an input group, zero bits are added (on the right) to form an integral number of 6-bit groups. Padding at the end of the data is performed using the "=" character.

Since all Base64 input is in 8 bit bytes, only the following cases at the end of the data stream can arise:

(a) the final block of input is 24 bits, therefore the final block of encoded output will be 4 characters (with no "=" padding).

(b) the final block of input is exactly 8 bits; here, zero bits are appended to produce a total of 12 bits. This final block is encoded as two characters followed by two "=" padding characters

(c) the final block of input is exactly 16 bits; here, zero bits are appended to produce a total of 18 bits. This final block is encoded output as three characters followed by one "=" padding character.

Example 8.1: To convert the text "*base64*" into Base64 encoding, from left to right split the characters into groups of 24 bits. The ASCII encoding of the characters and the binary equivalents are shown below

	First Group		
	b	a	s
Decimal	98	97	115
Hexadecimal	62	61	73
Binary	01100010	01100001	01110011

	Second Group		
	e	6	4
Decimal	101	54	52
Hexadecimal	65	36	34
Binary	01100101	00110110	00110100

Split the bytes into 6 bit words

	First Group			
6-bit	011000	100110	000101	110011
Decimal	24	38	5	51
Base64	Y	m	F	z

	Second Group			
6-bit	011001	010011	011000	110100
Decimal	25	19	24	52
Base64	Z	T	Y	0

So the characters "base64" become "YmFzZTY0".

Exercise 8.2: Under Linux there are a number of base64 utilities. If the package *MetaMail* is installed then you will have access to the Base64 encoding filter program `mimencode`. If not you can use the supplied code that can be found on the on the web site.

Using a Base64 program try the example above.

Using a Base64 program encode different length strings. When do the "=" characters appear at the end of the encoded string and why?

8.2 Content-type Header

It should be clear from the brief outline of Base64 encoding above, that unlike some encoding schemes Base64 does not encode any information about the encoded file into the output stream. This means that information must be sent along with the encoded file that describes the file's content. The MIME defines header fields to be added to mail messages that have encoded multimedia files attached. These header fields describe the contents of the encoded attachments.

The MIME header that describes an encoded document's contents, and also the header used by web servers and web clients is the content type header:

```
Content-Type:  type/subtype
```

The content type header describes the contents of the file to follow by referring to a standardised list of document types and subtypes.

The standard *media* document types for discrete documents are *text, image, audio, video, application*. There is also a *multipart* type that is used to describe a document that is made up of multiple documents of the same or different formats.

The *subtype* field specifies the *format* of the media document. That is, what sort of *text* file the document is, or what *audio* format the document is in. For example, a HTML document would have the content type of *text/html*, a text document in HTML format. A quicktime movie has the content type *video/quicktime*, a video document in Quicktime format.

Table 8.1 lists some of the more common MIME types.

You will notice from the table that some of the subtypes begin with the prefix **x-**. This prefix means that the subtype is experimental and has yet to be officially accepted[1].

MIME types and subtypes can be freely added by anyone, but until they are officially recognised they will not generally be known beyond the site where they are being used.

[1] See RFC2048, the fourth (of five) MIME RFC's. This one is on registration procedures of MIME types. Only look at it if you are really keen.

Type/Subtype	Extensions	Description
application/mac-binhex40	hqx	Macintosh Binhex 4.0 format
application/msword	doc	Microsoft Word Format
application/octet-stream	bin dms lha lzh exe class	Raw binary
application/postscript	ai eps ps	Adobe Postscipt
application/pdf	pdf	Adobe Portable Document Format
application/x-tar	tar	Unix tar archive
application/zip	zip	PKZip file compression format
application/x-gzip	gz	GNUzip compression format
application/x-httpd-php	php	PHP scripting language
audio/basic	snd au	Sun Microsystem's audio format
audio/x-wav	wav	Microsoft's 'wav' format
audio/x-realaudio	ra	Realaudio Format
image/gif	gif	Compuserve GIF format
image/jpeg	jpeg jpg	JPEG format
image/png	png	PNG format
text/html	html htm	Hypertext Markup Language
text/plain		plain text
text/css	css	Style Sheet
video/mpeg	mpeg mpg	MPEG movie format
video/quicktime	mov	Macintosh qicktime format
video/x-msvideo	avi	Microsoft video format

Table 8.1: Some common MIME types with their common file extensions.

8.3 Servers and MIME typing

When the server sends a requested document to the requesting client it precedes the document with a header (see §7.2.2) that includes (among other things) the MIME type of the document to be sent.

To be able to send the MIME type of the document the server must know what type of document has been requested. How does the server know the MIME type? There are a number of simple techniques that the server can employ to discern the document MIME type. To send a document the server must open it on the local system and read it. An obvious way for the server to learn the MIME type is to recognise it from the information in the file. Unfortunately there are many file types and new ones being created all the time. For every new file type the server would have to be rebuilt and code added to the server's parser or interpreter. A cumbersome procedure at best[2]. A simpler method would be if the server could recognise the MIME type from the file's URL.

[2] Apache has an optional module called "mime-magic" that can be built into the server. This module tries to match the first few bytes of a file against a table to find a file's MIME type.

File Type Extensions

For many years, operating systems (but not all) have been incorporating file type extensions in the file name. This was so the user could sort his or her files and recognise file types directly from the name. The standard that developed was to add a suffix to the file name of (at most) three letters to identify the file type. The suffix is delimited by a period. For example a text file would be recognised by the suffix *.txt*, a Compuserve GIF file by the suffix *.gif*, an Adobe postscript file with *.ps*. Originally file extensions were used so that people could recognise at a glance what was in a file without having to open the file. The web server behaves in exactly the same way. It uses the file extension to recognise the MIME type of the file.

To recognise the MIME type from the file extension, a server has a *look up table*. For each MIME type there is a list of the file extensions commonly used for that MIME type. The lookup table for the Apache server is loaded at startup from the file *mime.types* and can be found in your Apache configuration directory `~/CSC2406/conf`.

Exercise 8.3: Study the file `~/CSC2406/conf/mime.types` used by your apache server to resolve the MIME types of files.

8.4 Clients and MIME typing

When a client requests a document, it can preface the request with a list of preferred document types. If the server has several choices available to it for the requested document it can preferentially pick the format preferred by the client (see §7.1.2 on the HTTP header field Accept).

After the client has received a requested document it needs to make a decision about what to do with the document. The decision on how to dispose of the document depends on what sort of document has been received. By looking at the MIME type of the requested document the client can decide on the appropriate action.

Every browser has a number of document types that it can display natively, *i.e.* in its own windows. All browsers can display plain text; *text/plain*, and hypertext; *text/html*. Graphical browsers can display Compuserve GIF images; *image/gif*, JPEG format; *image/jpeg*, and some can display the PNG format; *image/png*. However there will always be formats that browsers can not handle, to deal with these formats the browser normally has a number of options

- Launch an external application and pass the job of interpreting the unknown format to the application. Such applications are called *Helper Applications*.
- Load an external module known as a *plug-in* to extend the browsers capabilities. The idea behind plug-ins is to keep the browser small (something that seems to be impossible). The capabilities of the plug-in are not required all the time only when a document of the correct MIME type is downloaded.

- Write the document to disk for processing later.
- Ask the user for help.
- Ignore the document.

Browsers make it easy to extend their capabilities to accommodate new document types.

Example 8.4: Suppose you have written a wonderful 3D visualisation application. The native files it produces can only be viewed by your program. How do you setup your server and client so that browsers can use your program as an external helper application automatically?

Client Side: Decide on a MIME type for your visualisation files, such as *application/x-my3d*
(Note this is an experimental MIME type!)

Add this mime type to your web browser and tell it the name of the helper application.

Server Side: Your wonderful 3D visualisation application produces files with the file extension `3dv`. The server needs to know which mime file type is represented by which local file extension. To do this the server has a table that maps the local file extension to the appropriate file type. The servers mime-type table needs to be modified so that the file extension `3dv` has mime type *application/x-my3d*.

Now when the client requests the file `image.3dv` the server knows that this file has mime type *application/x-my3d* which will be incorporated into the header information by the server as

```
Content-type:  application/x-my3d
```

When the client parses the server header information it will know to launch the helper application specified for this mime type.

Exercise 8.5: Set up your server and client to be able to recognise your personal file types. Create a text file using your favourite editor. Give it a unique file extension, for example `mft`. Place the file into the server document tree. Decide on a MIME type for your new file format, for example `x-myfiletype`. Edit the Apache *mime.types* file (found in the `~/CSC2406/conf` directory) to include your new file extension and mime type. Now set up your client to launch a helper application when the server notifies the client of your new MIME type. Use a simple text editor/reader as the helper application.

Now create a page that has a hyper-link to your new file. When the file is requested from your client the helper application should be launched to view your new file.

8.5 Questions

Short Answer Questions

Q. 8.6: Why was base64 encoding developed?

Q. 8.7: Why is the common mail encoding scheme called *base64*?

Q. 8.8: What is the "=" character used for in base64 encoding?

Q. 8.9: Explain, in broad terms, what information the MIME header *Content-type* contains.

Q. 8.10: Why does the content type field require a *type* and *subtype*?

Q. 8.11: How is the file extension used in MIME typing?

Q. 8.12: Why does the Web server need to send the MIME type of the document to the client?

Q. 8.13: What are *Helper applications*?

Q. 8.14: What are *plug-ins*?

Q. 8.15: How are new MIME types defined for the server?

Q. 8.16: What needs to be changed or added to the server and client when a new MIME type is created?

8.6 Further Reading and References

The following RFC's are uncommonly dull, even by the standard of RFC's, be warned!

- The Apache server mime types can be found in the file

  `~/CSC2406/httpd/conf/mime.types`

  in your distribution of the Apache server.
- www.hunnysoft.com/mime/ MIME information page
- RFC2045 *Multipurpose Internet Mail Extensions* (MIME) Part One: Format of Internet Message Bodies
- RFC2046 *Multipurpose Internet Mail Extensions* (MIME) Part Two: Media Types
- RFC2047 *Multipurpose Internet Mail Extensions* (MIME) Part Three: Message Header Extensions for Non-ASCII Text
- RFC2048 *Multipurpose Internet Mail Extensions* (MIME) Part Four: Registration Procedures
- RFC2049 *Multipurpose Internet Mail Extensions* (MIME) Part Five: Conformance Criteria and Examples

Chapter 9 HTML Forms

The FORM tag specifies a fill-out form within an HTML document. The tag provides a method of requesting information from a web client.

Chapter contents

9.1 The Form element

Attributes: action, method, enctype

The form tag specifies a fill out form within an HTML document. The tag provides a method of requesting information from a web client.

Example 9.1: An example of a fill out form that requests information from the client.

An HTML document can contain multiple `form` tags but they **cannot** be nested.

Within the `form` container there can be specified all the standard HTML elements plus several kinds of form fields such as single and multi-line text fields, radio button groups, checkboxes, and menus.

The attributes of the form tag are:

`action`

This specifies a URL which is to *somehow* handle the information entered in the form. For example, to post the information in the form via email,

`action="mailto:lec.CSC2406@usq.edu.au"`,

or to invoke an application via HTTP (see the CGI scripts module 10),

`action="http://www.sci.usq.edu.au/register.php"`

`method`

When the action attribute specifies an HTTP server, that is,

`action="http://www.sci.usq.edu.au/staff/"`

the method attribute determines which HTTP method will be used to send the form's contents to the server. It can be either `GET` or `POST`, and defaults to `GET`. The method attribute chosen determines how the server and the forms handling application will receive the information entered into the form (see the CGI scripts module 10).

`enctype`

This determines the mechanism used to encode the form's contents and submit it to the server. This is used when the `method` is `POST`. It defaults to the MIME type `application/x-www-form-urlencoded`. The other form of encoding is `multipart/form-data` which must be specified if the `input` element, `type="file"` has been used.

9.2 Form Input elements

There are three types of data input elements that can only be found within a `form` container: `input`, `textarea`, and `select`. The `input` element can be used for a variety of form fields including single line text fields, password fields, checkboxes, radio buttons, submit and reset buttons, hidden fields, file upload, and image buttons. The `select` element is used for single or multiple choice menus. The `textarea` element is used to define multi-line text fields.

9.2.1 `<INPUT TYPE="TEXT"...>`

Attributes: `name` (required), `value`, `size`, `maxlength`

A single line text field whose visible size can be set using the `size` attribute, e.g. `size=""40"` for a 40 character wide field. Users should be able to type more than this limit with the text scrolling through the

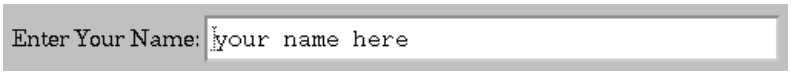
field to keep the input cursor in view. An upper limit can be enforced on the number of characters that can be entered with the `maxlength` attribute. The `name` attribute is used to name the field, while the `value` attribute can be used to initialise the text string shown in the field when the document is first loaded.

Example 9.2: An example of using the `type="text"` input element.

The HTML code

```
Enter Your Name:
<input type="text"
       size="40"
       name="user"
       value="your name here" />
```

is rendered in the following way:



9.2.2 <INPUT TYPE="PASSWORD"...>

Attributes: `name` (required), `value`, `size`, `maxlength`

This is similar to the `type="text"` input element, but obscures the input text by echoing a character like `*`. When entering a password this is required to hide the text from prying eyes.

You can use `size` and `maxlength` attributes to control the visible and maximum length exactly as the regular text fields.

Example 9.3: An example of using the `type="password"` input element.

```
Enter Your Password:
<input type="password"
       size="12"
       name="pw" />
```

This is rendered exactly the same as the `type="text"` element. See Example 9.2

9.2.3 <INPUT TYPE="CHECKBOX"...>

Attributes: `name` (required), `value`, `checked`

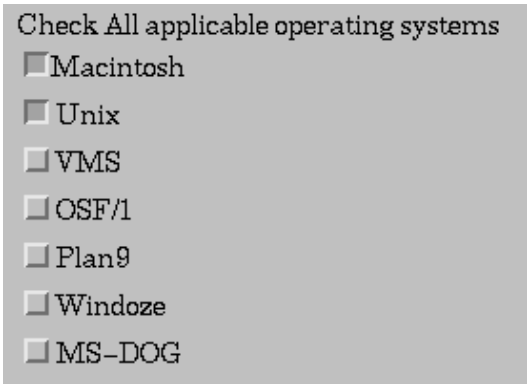
This element is used for simple Boolean fields, or for fields that can take multiple values at the same time. The latter is represented by several checkbox fields with the same `name` and different `value` attributes. Each checked checkbox generates a separate name/value pair in the submitted data, even if this results in duplicate names. Use the `checked` attribute to initialise the checkbox to its checked state.

Example 9.4: An example of using the `type="checkbox"` input element.

The HTML code

```
<form>
Check All applicable operating systems<br/>
<input type="checkbox" name="os"
       checked="checked"
       value="macintosh"/> Macintosh<br/>
<input type="checkbox" NAME="os"
       checked="checked"
       value="unix"/> Unix<br/>
<input type="checkbox" name="os"
       value="vms"/> VMS<br/>
<input type="checkbox" name="os"
       value="osf1"/> OSF/1<br/>
<input type="checkbox" name="os"
       value="plan9"/> Plan9<br/>
<input type="checkbox" name="os"
       value="doze"/> Windoze<br/>
<input type="checkbox" name="os"
       value="dos"/> MS-DOG<br/>
</form>
```

is rendered in the following way:



9.2.4 <INPUT TYPE="RADIO"...>

Attributes: `name` (required), `value` (required), `checked`

This element is used for fields which can take a **single** value from a set of alternatives (cf. `checkbox`). Each radio button field in the group should be given the same `name`. Radio buttons require an explicit `value` attribute. Only the checked radio button in the group generates a name/value pair in the submitted data. One radio button in each group should be initially checked using the `checked` attribute.

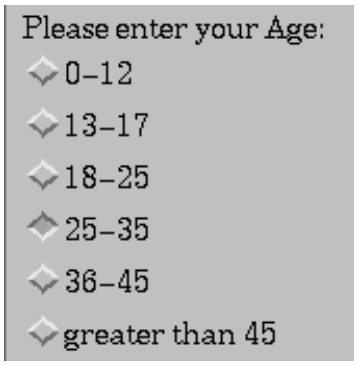
Example 9.5: An example of using the `type="radio"` input element.

The HTML code

```
<form>
Please enter your Age:<br/>
<input type="radio" name="age"
       value="12"/> 0-12 <br/>
<input type="radio" name="age"
       value="17"/> 13-17<br/>
```

```
<input type="radio" name="age"
       value="25"/> 18-25<br/>
<input type="radio" name="age"
       value="35"
       checked="checked"/> 25-35<br/>
<input type="radio" name="age"
       value="45"/> 36-45<br/>
<input type="radio" name="age"
       value="gt"/> greater than 45<br/>
</form>
```

is rendered in the following way:



9.2.5 <INPUT TYPE="SUBMIT"...>

Attributes: `name`, `value`

This element defines a button that users can click to submit the form's contents to the server. The button's label is set by the `value` attribute. If the `name` attribute is given then the submit button's name/value pair will be included in the submitted data. You can include several submit buttons in the form. See `type="image"` for a graphical submit button.

Example 9.6: An example of using the `type="submit"` input element.

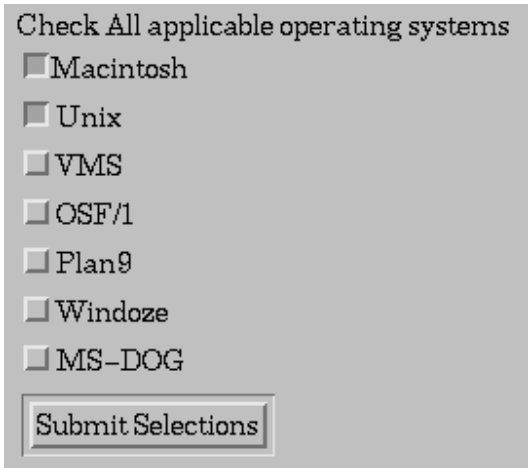
The HTML code

```
<form>
Check All applicable operating systems<br/>
<input type="checkbox" name="os"
       checked="checked"
       value="macintosh"/> Macintosh<br/>
<input type="checkbox" name="os"
       checked="checked"
       value="unix"/> Unix<br/>
<input type="checkbox" name="os"
       value="vms"/> VMS<br/>
<input type="checkbox" name="os"
       value="osf1"/> OSF/1<br/>
<input type="checkbox" name="os"
       value="plan9"/> Plan9<br/>
<input type="checkbox" name="os"
       value="doze"/> Windoze<br/>
```

```
<input type="checkbox" name="os"
       value="dos"/> MS-DOG<br/>

<input type="submit" name="send"
       value="Submit Selections"/>
</form>
```

is rendered in the following way:



9.2.6 <INPUT TYPE="IMAGE"...>

Attributes: `name` (required), `src` (required), `align`

This element is used for graphical submit buttons rendered by an image rather than a text string. The URL for the image is specified with the `src` attribute. The image alignment can be specified with the `align` attribute.

The coordinates of the location clicked are passed to the server. In the submitted data, image fields are included as two name/value pairs. The name/value pairs are `name.x=x-value` and `name.y=y-value`. Where `name` is the value of the `name` attribute. The `x-value` is measured in pixels from the left edge of the image, and the `y-value` is measured in pixels from the top edge of the image.

Note

If the server takes different actions depending on the location clicked, users of non-graphical browsers will be disadvantaged. For this reason, this element should be used with caution, and alternate methods should be considered first.

9.2.7 <INPUT TYPE="RESET"...>

Attributes: `name`, `value`

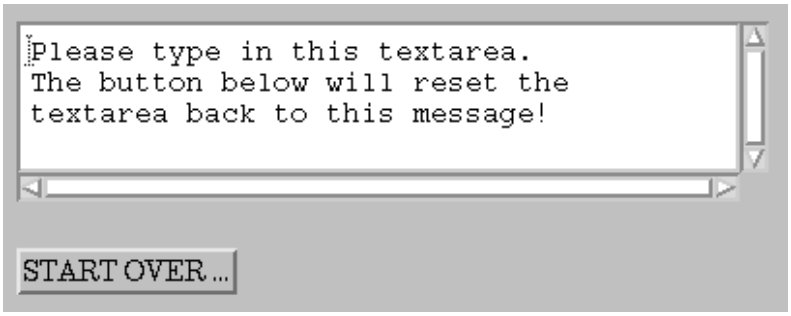
This element defines a button that users can click to reset form fields to their initial state. The reset button label can be set by providing a `value` attribute. Reset buttons are never sent to the server as part of the form's contents.

Example 9.7: An example of using the `type="reset"` input element.

The HTML code

```
<form>
<textarea cols="40" rows="4">
Please type in this textarea.
The button below will reset the
textarea back to this message!
</textarea>
<p>
<input type="reset" value="START OVER ..."/>
</p>
</form>
```

is rendered in the following way:



9.2.8 `<INPUT TYPE="FILE"...>`

Attributes: `name` (required), `size`, `maxlength`, `accept`

This element provides a means for users to attach a file to the form's contents. It is generally rendered by a text field and an associated button which when clicked invokes a file browser to select a file name. The file name can also be entered directly in the text field. Just like the `type="text"` element the `size` attribute can be used to set the visible width of this field in character widths. The upper limit to the length of file names can be set using the `maxlength` attribute. Some web clients support the ability to restrict the kinds of files to those matching a comma separated list of MIME content types given with the `accept` attribute e.g. `accept="image/*"` restricts files to images.

Note

To be able to include the contents of a file with a form submission the `enctype` attribute of the form must be set to `enctype="multipart/form-data`

Further information can be found in RFC1867 and the HTML4.0 specification.

Example 9.8: An example of using the `type="file"` input element.

The HTML code

```
<form enctype="multipart/form-data" method="POST">
Please select your photo file<br/>
```

```
<input type="file" name="photo" size="40" accept="image/*">
</form>
```

is rendered as:

9.2.9 <INPUT TYPE="HIDDEN"...>

Attributes: `name` (required), `value`

This element is **not** rendered and provides a means for servers to store state information with a form. That is, HTTP connections do not maintain a history of previous connections, so it is difficult to carry on a `conversation` when context matters. The `type="hidden"` can be used to store the history of previous connections.

The name/value pair is passed back to the server with the rest of the form data.

Note

As there is no reliable way to *hide* the HTML that generates a page, the `type="hidden"` element should **not** be used to embed sensitive information you do not want the client to see.

Example 9.9: An example of using the `type="hidden"` input element.

The HTML code

```
<form>
<input type="hidden" name="LastAction" value="delete''>
</form>
```

will not be rendered but the name/value pair `LastAction=delete` will be returned to the server.

Exercise 9.10: To gauge customer satisfaction, an online business plans to place a web page on their site, that among other things asks customers to fill out a questionare.

Using as many of the input fields listed above design a web page that could be used for customer input.

The choice of business is up to you.

9.3 SELECT element

Attributes: (container), `name` (required), `size`, `multiple`

The `select` element is used to present a set of options to the user. If only a single entry can be selected and no visible size has been specified, the options are typically presented in a drop down menu. If multiple selections are permitted or a specific visible size has been specified then the options are presented as a list box.

The attribute `name` identifies the selection. The `size` attribute defines the number of visible rows to be displayed when rendering the list. The `multiple` attribute specifies that multiple entries can be selected from the list. If `multiple` is omitted, only single selection is permitted

9.3.1 <OPTION...>

Attributes: `value` , `selected`

The `option` element is only valid within a `select` element, and specifies the menu choices.

The `value` attribute gives the value to be returned if this item is selected. It is **not** the text to be displayed for this option; that is specified by the text that is listed *after* the `option` tag.

If present, the `selected` attribute specifies that the associated menu item will be selected when the page is first rendered. It is an error to specify more than one selected item if the multiple selections are not permitted.

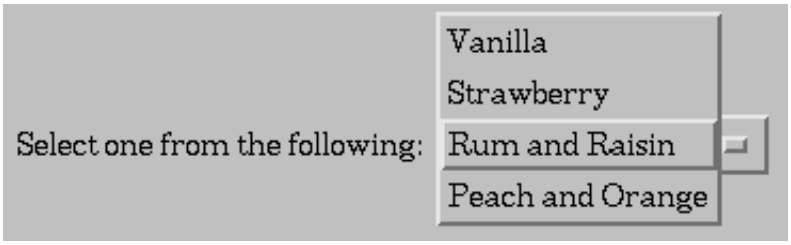
Example 9.11: An example of using the `SELECT` input element with only a single selection allowed.

The HTML code

```
<form>
Select one from the following:
<select name="flavour">
    <option value="a"> Vanilla </option>
    <option value="b"> Strawberry </option>
    <option value="c"> Rum and Raisin </option>
    <option value="d"> Peach and Orange </option>
</select>
</form>
```

is rendered as:

and when clicked on becomes

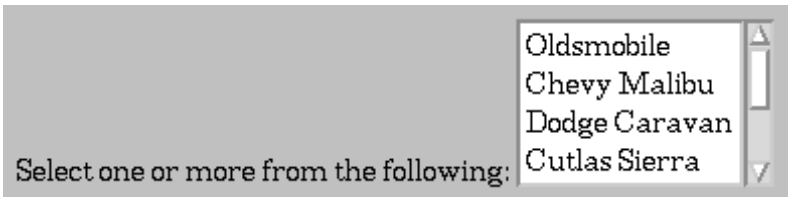


Example 9.12: An example of using the `select` input element with multiple selections allowed.

The HTML code

```
<form>
Select one or more from the following:
<select name="model" size="4" multiple="multiple">
    <option value="a" />Oldsmobile </option>
    <option value="b" />Chevy Malibu </option>
    <option value="c" />Dodge Caravan </option>
    <option value="d" />Cutlass Sierra </option>
    <option value="e" />Pickup Truck </option>
    <option value="f" />Sunbird </option>
    <option value="g" />Miata </option>
</select>
</form>
```

is rendered as:



9.4 TEXTAREA element

Attributes: `name` (required), `rows` (required), `cols` (required), `wrap`

This element is used to input *multiline* text fields. The content of the element is restricted to text and character only. The character set for submitted data should be ISO Latin-1, unless the server has previously indicated that it can support alternative character sets.

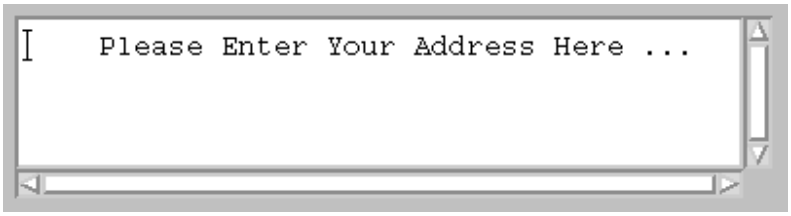
The `rows` attribute specifies the number of *visible* rows, not the maximum number of rows. The `cols` attribute specifies the with that is *visible* (in units of average character widths). If the user requires more rows or columns that is visible the web client *should* provide scroll bars.

Example 9.13: An example of using the `textarea` input element.

The HTML code

```
<form>
  <textarea name=address rows="4" cols="40">
  Please Enter Your Address Here ...
  </textarea>
</form>
```

is rendered as:



Ex. 9.14: Expand exercise 9.10 to allow comments to be submitted by customers through a `textarea` tag

Ex. 9.15: Create a simple forms page (that is, do not use any of the `form` element attributes). Only include a submit button within the `form` container (See section 9.2.5).

When the submit button is pressed note the URL that appears in the `Location` field of the browser. Add an input text line to the form. Again press the submit button. Again note the URL being sent to the server. Change the form by changing the type of input fields. Each time the submit button is pressed note the URL sent to the server.

Explain what is happening.

What do you think the "+" characters represent in the URL (if there are any)?

What do you think the "?" and "&" characters are for?

Ex. 9.16: Repeat the exercise above but with two `form` containers on the one page page. Each container should contain a submit button. Experiment with giving the buttons different name/value pairs.

Explain how different submit buttons with different names or values can be useful.

9.5 Form Elements and CSS

CSS2 did not define style commands for HTML form elements— most browsers though recognise style commands for form elements. Since there is no standard care must be taken however as elements can look very different in different browsers.

For forms as is the case with many HTML elements experimentation with styling commands is required.

9.6 Questions

Short Answer Questions

Q. 9.17: What is a form in a HTML document?

Q. 9.18: The HTML `forms` tag has a number of attributes. Explain the `action` and `method` attributes.

Q. 9.19: If the `action` attribute is not set, where is the forms data sent?

Q. 9.20: How does the user tell the client to send the contents of the form to the server?

Q. 9.21: How are the contents of the form sent to the server?

Q. 9.22: Explain why *clients* need to know about MIME types and file suffixes?

Q. 9.23: Why have *hidden* fields in a form?

9.7 Further Reading and References

(a) The W3C HTML4.0 standard defines the Forms elements in detail including elements and attributes not discussed in this module.

Chapter 10 Server Scripts and the Common Gateway Interface

Chapter contents

10.1 Introduction

One of the most powerful features of the World Wide Web is the ability of the Web server to execute programs that can create dynamic documents. The programs can be as simple or as complex as you like. You can use them as document word-search engines, as interfaces for controlling external machinery, as electronic order forms, or as gateways to other information services.

These programs, that are external to the web server, are called *server scripts*, and can be written in any language. Some are written in an interpreted language such as PHP (the language most favoured for server scripts), Perl, Python or in a compiled language such as C, C++,.... The choice of language is entirely up to the script writer.

When a user requests a URL that points to a script, the server executes the script. Any output the script produces is returned, by the server, to the user's browser for display. URLs used to invoke server scripts look just like any other URL.

Note

PHP scripts behave subtly differently in that they are not run as a separate executable—because the PHP interpreter is built into Apache as a module.

Even though the PHP module is built-in in many ways it behaves as a separate executable.

Exercise 10.1: Your Apache server has a number of server scripts included. An example of a server script can be found at URL (insert your hostname and port number in place of `hostname:port` in the following URLs)

`http://hostname:port/cgi-bin/`

Have a look in the directory `~/CSC2406/cgi-bin` and the examples of server scripts in the source directory `~/CSC2406/src`

10.2 Script Identification

How does the server know that the resource the URL points to is to be executed and not the name of a file to be returned to the browser for display? There are two alternate methods used. One method is to designate particular directories on the web site as server script directories. By default the Apache server defines one directory in the server root, called *cgi-bin* to contain server scripts. The server treats every file in this directory (or subdirectories below it) as an executable script not a regular document to be returned to the browser.

The second method for identifying scripts, used by the Apache server and some others, is to use a filename extension usually *.cgi*, to identify server scripts. Server scripts identified in this way do not have to live in a particular directory, they can reside anywhere in the document tree. If both methods of identifying server scripts are available they can be intermixed, placing some scripts in script directories and others elsewhere in the document tree.

Note

PHP scripts normally run using the second method. Your server has been configured to recognise the "`php`" extension as a PHP script.

10.3 Communicating with Scripts

Some scripts don't require user input. You call them, via their URL, and they display something. Scripts that don't require input have limited functionality. The interesting scripts require additional information from the user: instructions to control a robotic arm, list of e-mail addresses to which to send a message, a person's name to look up in a phone data base.

Scripts that require extra information will usually create documents that give the user the opportunity to fill out a form. Alternatively you can provide the script with the information it requires directly - through the URL.

10.3.1 Passing Parameters

The most direct way to send information to a script is to add the information to the URL. To do this add a "**?**" (question mark) to the end of the script's URL, everything following the question mark is passed to the script. The string following a question mark is called the "*query string*".

Example 10.2: The following snippet of HTML demonstrates passing a string to a script

```
<a href="/scripts/find.php?Kubla%20Khan"><br>
In Xanadu did Kubla Khan<br>
A stately pleasure-dome decree;<br>
...<br>
<a>
```

The URL of the script is "`/scripts/find.php`", the string being passed to the script is "`Kubla Khan`".

Like other parts of the URL, spaces, tabs, carriage returns and other reserved characters must be escaped using the "`%`" character followed by the 2-digit hexadecimal code[1].

So the space between `Kubla` and `Khan` becomes `%20`

Exercise 10.3: A simple script supplied with your version of the Apache server is `http.php`. This script will echo back all the information it is given by the server. The URL for this script should be

```
http://hostname:port/http.php
```

Pass strings to this script from your browser by appending the strings to the URL. The strings should appear in the `Query_String` environment variable. Use escaped characters in your strings. Are the returned values still escaped? What does this mean for the author of server scripts?

Although a script is free to use any format for the query string, in practice query strings fall into two predefined categories, *keyword lists* and *named parameter lists*.

Keyword List

This list is most often used by scripts that perform word searches and is used by the web browser when an `ISINDEX` HTML element has been used in a document.

This type of query string is made up of a series of phrases separated by "`+`" signs. The `+` sign separates the phrases. Phrases can be made up of more than one word. To separate words within a phrase the normal URL escape sequence `%20` for a space is used.

Example 10.4: An example of a keyword list is

[1] For all the character codes see the character codes on the course web site.

```
<a href="/scripts/find.php?Samuel+Taylor+Coleridge">
The Rime of the Ancient Mariner
<a>
```

Exercise 10.5: Use Example 10.4 to pass keyword lists to the script "test".

For example
`http://hostname:port/http.php?Samuel+Taylor+Coleridge`

This form of a query string was developed originally for the `ISINDEX` HTML element.

Named Parameter List

The keyword list has limited extensibility. A method of passing name-value pairs allows for greater flexibility in the information passed to a script. To this end the named parameter list was devised. A named parameter list has the form

`name1=value1&name2=value2&name3=value3&...`

It consists of a series of `name=value` pairs separated by an ampersand "&". Each pair defines a named parameter for the script to use.

Example 10.6: An example of name-value list is

```
<a href="/scripts/find?author=Coleridge&poem=Christabel">
... The lovely lady, Christabel ...
<a>
```

This example passes two parameters to the script "`/cgi-bin/find`". The first parameter, "`author`" has the value "`Coleridge`", the second parameter, "`poem`" has the value "`Christabel`".

Exercise 10.7: Use the example above to pass parameter lists to the script "test". For example (without the break in the URL)

```
http://hostname:port/
            http.php?name=John+Smith&id=q9876543X
```

Note: the `+` symbol, as well as `%20` can be used to escape the space in parameter lists)

To pass arguments to a script that uses parameter lists, you need to know what parameter names the script expects. The names a script expects are normally a function of the script task, e.g. a library catalogue search script might expect arguments named *author*, *title*, *keywords* etc. Usually this information is given in the script documentation.

10.3.2 Passing Path Information

Scripts can be passed additional path information instead of, or in addition to, regular query strings. The path is usually the partial URL of some document elsewhere on your site. What the script does with the partial URL is entirely script dependent. For example, the partial

URL might point to a configuration file that can modify the behaviour of the script, or the script could parse the document modifying its contents before passing it back to the requesting browser.

To send additional path information to a script that expects it, just append the path on to the end of the script's URL.

Example 10.8: To pass the document URL `/coleridge/christabel.txt` to the script `http://hostname:port/cgi-bin/formatpoem`, the full URL becomes (without the break)

```
http://hostname:port/
          cgi-bin/formatpoem/coleridge/christabel.txt
```

The server processes the URL element by element. Once it recognises the name of an executable script it executes the script and passes the remainder of the URL to the script.

The advantage of this method is that documents that need processing by server scripts have URL's that are not overly complicated.

Exercise 10.9: Pass path information to the script *http.php*. For example,

```
http://hostname:port/http.php/test/path/info
```

What effect (if any) does adding parameters to the URL have? For example (without the break in the URL),

```
http://hostname:port/
    http.php/path/info?name1=value1&name2=value2
```

10.3.3 HTML Input

Though you can send information to scripts by including parameters explicitly in the URL, this approach is limited - you have to remember the parameters of every script you might wish to utilise and the way the script expects to receive them. A more powerful way to send data to scripts is to create input documents, HTML pages that contain input fields for the user to fill out and then submit to the script.

Keywords Input

Keyword input is the older form of input HTML documents. It was used when scripts where mainly text-search engines. The HTML element `ISINDEX` is used for this form of keyword searches.

Exercise 10.10: Create an HTML document in your document tree containing the following HTML fragment

```
Enter keywords here:<br>
<ISINDEX ACTION="/http.php">
```

Access the document and enter some keywords and press the return key.

What is the result? What do you think the client has done?

The `ISINDEX` element creates a text entry point for the user to enter keywords. When the return key is pressed the browser bundles up the keywords into a *keyword style query string* and sends them to the script.

This interface is used by older scripts, by simple scripts, and in cases where the script's author wanted to maintain compatibility with old browsers that don't have fill-out forms.

Forms Input

In contrast to the keyword input, Forms based input is much more flexible. With forms you can define text fields, checkboxes, radio buttons, pop-up menus, and scrolling lists. When the user fills out the form and presses the *submit* button, the browser bundles the current contents of the form into a *parameter list-style query string* and sends it to the server which passes it to the script.

Example 10.11:

An example of a form is the following:

```
<FORM ACTION="/http.php"
      METHOD="POST"
      ENCTYPE="x-www-form-urlencoded">
<INPUT TYPE="RADIO" NAME="group"
       VALUE="1" CHECKED> Bitter
<INPUT TYPE="RADIO" NAME="group"
       VALUE="2"> Pale Ale
<INPUT TYPE="RADIO" NAME="group"
       VALUE="3"> Porter
<INPUT TYPE="RADIO" NAME="group"
       VALUE="4"> Stout
</FORM>
```

NAME Attribute How does the browser know how to bundle the information from the forms page into name/value pairs? All form input tags have a `NAME` attribute with an assigned value (this attribute is mandatory for all fields). The person designing the forms page and the server script decides on the name to be attached to each input field. When the forms submit button is pressed the browser bundles each input field into name/value pairs using the value of the `NAME` attribute and the current value of the input field.

ACTION Attribute Where is the form information sent when the submit button is pressed? One of the attributes of the `FORM` tag is the `ACTION` attribute. The value of this attribute is the URL of the script that should receive the name/value pairs from the form page. If the `ACTION` attribute is missing the client will send the Form data to the current URL.

ENCTYPE Attribute Another of the attributes of the `FORM` tag is the `ENCTYPE` attribute. This attribute tells the client how to bundle up the information in the forms into name/value pairs. The default encoding type is

`x-www-form-urlencoded`.

Currently the only other encoding type is

`multipart/form-data`,

which is the mandatory type if the `TYPE="FILE"` input element is used. The methods outlined above for passing information to scripts with special characters escaped is the default URL encoding scheme. It is safe to leave this attribute out.

METHOD Attribute The `METHOD` attribute of the `FORM` tag, tells the client how to pass the information from the form to the server and tells the script how the information from the form will be received. The two methods we will be concerned with is the `GET` and the `POST` methods. The default method is `GET`.

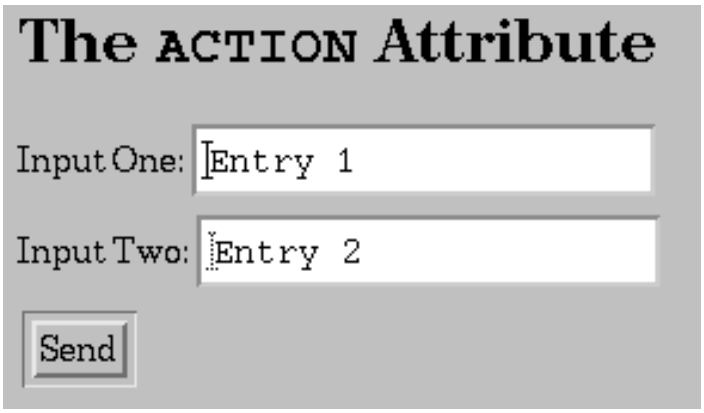
When the `GET` method is used the data is appended to the end of the URL as discussed above. When the `POST` method is used the data is sent to the server after the header fields. This means that the `POST` method is better able to send large amounts of data to the server.

Example 10.12: Consider the following HTML page

```
<!DOCTYPE HTML PUBLIC
      "-//W3C//DTD HTML 4.01 Transitional//EN"
      "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
 <title>The ACTION & METHOD attribute</title>
</head>
<body>
 <h1>The <code>ACTION</code> Attribute</h1>
 <form
    action="http://localhost:9000/http.php"
    method="GET">
  Input One:
  <input type="TEXT"
         name="TextField1"
         value="Entry 1">
  <br>
  Input Two:
  <input type="TEXT"
         name="TextField2"
         value="Entry 2">
  <br>
  <input type="SUBMIT"
         name="SubmitButton"
         value="Send">
 </form>
```

```
</body>
</html>
```

When rendered by the web client the page looks like this



When the submit button is pressed the *server* is sent the following data by the client

```
GET /http.php?TextField1=Entry+1&TextField2=Entry+2&
    SubmitButton=Send HTTP/1.0
Connection: Keep-Alive
User-Agent: Firefox/3.0 [en]
Host: localhost:9000
Accept: image/gif, image/jpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
```

This shows that the `GET` method, the default method, attaches the name/value pairs of the forms page to the URL defined by the `ACTION` method.

When the `POST` method is used the same page is sent to the server in the following way

```
POST /http.php HTTP/1.0
Connection: Keep-Alive
User-Agent: Firefox/3.0 [en]
Host: localhost:9000
Accept: image/gif, image/jpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Content-type: application/x-www-form-urlencoded
Content-length: 55

TextField1=Entry+1&TextField2=Entry+2&SubmitButton=Send
```

The forms data is now sent as data following the headers. The `Content-type` header tells the server how the data is encoded

and the `Content-length` header tells the server how much data will follow the header fields.

10.4 Communicating with Clients

In the previous sections we have seen how the browser communicates information to the server, but how does the script communicate to the browser? We also need to discover, in later sections, how the script receives the browser information from the server.

To communicate information to the browser the script only has to write to *standard out*. For example the "`printf`" command in C will print a string to standard out. The server sends all output from the script to the browser.

To conform to the HTTP the server needs to add the mandatory headers. One header the server cannot add is the *Content-type* header[2]

10.4.1 Content-type

In Module 8, on MIME typing we discovered that the server used the HTTP header element `Content-type` to inform the browser the type of document to follow. The server decides on the MIME type to assign to the document by the filename extension of the document. Server scripts can have any file extension and can return any sort of document, in fact a script can return different document types with each invocation. So how does the server or the browser know what sort of document to expect from a server script?

The script itself must supply the `Content-type` header so that the browser knows the type of data that will follow. The server will supply the minimum required header fields and then will append the output from the script.

It is the responsibility of the script to supply the `Content-type` field. Without this field the output from the script will not be displayed correctly by the browser. The server will also check for correctly formed output and will replace the script output with an error message to the browser.

Note

As mentioned in Module 7, on the HTTP, a correctly formatted header is terminated with a blank line. The scripts header fields are appended to the servers, so the script **must** supply the blank line to terminate the header after it has sent it's header fields.

Technically the protocol calls for the header lines to be terminated with a carriage return/linefeed pair. In practice most servers are flexible on this point and will accept a single new line character as a line terminator.

[2] If you want your script to bypass the header inclusion mechanism of the server then by adding the string `nph-` (No Parse Header) to the script name the Apache server will connect the output of the script directly to the input of the browser. This means that the script **must** act like a server and send all the appropriate HTTP headers. The server has stepped aside and left all the communication up to the script.

Note

PHP assumes a `Content-type` of `text/html`—so that PHP scripts normally do not have to supply the HTTP header `Content-type`. one the other hand if the script is not outputting `text/html` then the script *must* supply the header line *before* any other output:

```
header('Content-type:  application/pdf');
```

The function `header()` outputs a header line and must be called before any other output from the scripts.

Exercise 10.13: The following simple C program will create an HTML document when run as a server script. Compile the program and run it from your server. Use the `view source` menu item on your browser to check the source received by your browser.

Experiment with alternate HTML code.

```
#include <stdlib.h>
#include <stdio.h>

void main(int argc, char *argv[], char *env[])
{
/*
** Print out the protocol header line
** and terminate with a blank line
*/
  printf("Content-type: text/html\n\n");
/*
** print out the HTML header lines
*/
  printf("<html><head>\n");
  printf("<title>Simple Script</title>\n");
  printf("</head><body>\n");
/*
** Here is the body of the document
*/
  printf("<h1>Simple Script with Output</h1><p>\n");
  printf("A simple message from a simple script!\n");
/*
** Finish the document so we have well-formed HTML
*/
  printf("</body></html>\n");

}
```

Exercise 10.14: The following simple PHP program will output an JPEG image. Place the script in a directory containing JPEG images and run it via your server.

Add comments to every line of the script.

```
<?php
$files=glob("*.jpg") or
 exit("Failed to find any images");
```

```
$index=rand(0,count($files)-1);

header('Content-type: image/jpg');

readfile($files[$index]);

?>
```

10.4.2 Location

Sometimes you don't want a script to create a document when it is run but to choose intelligently among a number of different URLs to send back to the browser. For example, you might wish to display one document to local users and another to visitors from remote locations. If this is the case then instead of printing a *Content-type* field in the header the script should print a *Location* field pointing to the URL you want the browser to fetch. When the server sees this field in the header information from the script, it generates a redirection directive to the browser which silently fetches the new URL. As there is no document to produce, the script need only print the blank line after the header information and exit.

Exercise 10.15: Write a simple script that will randomly select a file from a directory (such as a directory of images). Pass the URL of the randomly selected file pack to the server using the *Location* header field. Do not return the file itself.

Use telnet to talk to the server and use the `GET` method to request this script. How is the server translating the script *Location* header so that the browser gets the correct document?

10.4.3 Dynamic Documents

So far we have discussed how the user can manually pass information to the script through the URL parameter list or through an input document of fill-out forms. The disadvantage of these methods are that they require you to know the *names* the script is expecting and requires the maintenance of multiple documents. As we have seen above the script output can be used to produce well formatted HTML that is sent to the browser by the server for display. This means that a script should (and in most cases does) create its *own* input documents on the fly. The standard design for server scripts is as follows:

(a) The browser requests the script *without* any parameters.

(b) The script is run by the server and because it has nothing to work with (no parameters) it builds its own input document and returns it to the browser.

(c) The user fills out the input document and sends it back to the server by pressing the *submit* button.

(d) The server calls the script again, but this time the contents of the fill out document are sent to the script as parameters.

(e) With the input sent from the user the script now has something to do. The script processes the users input and sends the results back to be displayed on the user's browser.

The script builds its own input form and formats the results.

If you are not satisfied with a script's built-in user interface, or if you wish to customise it with site-specific information (see Section 10.7 on state information) you can create your own custom front-end for the script by writing an input document for the script. Your custom input document could be a forms page with the `ACTION` attribute set to the script you are not satisfied with.

10.5 Common Gateway Interface (CGI)

So far we have discussed how the web client bundles up the information it wishes to send to the server, and we have discussed how the script talks to the web client, but we have not discussed how the server sends the information from the client to the script.

The Common Gateway Interface (CGI) defines how the server and the script communicate. It defines where the script will find the various bits of information that has been encoded and sent from the browser.

Unfortunately, the interface between server and script went through several evolutionary phases as the demands on scripts became more complex. The mechanisms that the server uses to send information to scripts can therefore seem arbitrary and redundant as backward compatibility is maintained. In some cases the same information is presented to the script in a number of different ways. Which version of the information the script uses is entirely up to the script.

Where the information is found depends on the type of information sent to the script, such as *path information*,*named parameter list*, *keyword list* and the *method* used to invoke the script, *e.g.* `GET` or `POST`. Currently these are the two most important methods for invoking server scripts.

10.5.1 Environment Variables

Before the server invokes the script, it fills the script environment with useful information about itself, the current request, and the remote browser. The information is stored in "environment variables"[3] that the server will set before invoking the script.

[3] Environment variables are variables that are maintained by the operating system. They are stored as *name/value* pairs and maintain a list of useful information that programs can access to learn about the environment they are running under. On a Unix box, at the prompt (under the C-shell and BASH) the command `printenv` will list all the defined environment variables. In C programs environment variables are accessed through the parameter list of the `main` routine. For example `int main(int argc, char *argv[], char *env[])` The optional string array `char *env[]` contains the environment variables.

In PHP scripts they can be found in the associative array `$_ENV` Under most operating systems all languages have the ability to access environment variables.

The following is a summary of *some* of the CGI environment variables (For a complete list of environment variables set by the Apache server, see the Apache documentation).

SERVER_SOFTWARE	The name and version number of the server software *e.g.* "Apache/1.3.41".
GATEWAY_INTERFACE	The gateway interface and version number, currently "CGI/1.1".
SERVER_PROTOCOL	The server protocol and version number used to invoke the script; either "HTTP/1.0" or "HTTP/1.1".
SERVER_NAME	Server's host name, *e.g.* `www.sci.usq.edu.au`.
SERVER_PORT	The port on which the server is listening.
SCRIPT_NAME	The name and path of the script executing.
AUTH_TYPE	Authorization type associated with the URL that called the script. For example "Basic".
REMOTE_USER	Name of the remote user when using username/password authentication.
REMOTE_HOST	The fully qualified domain name of the remote host, if known.
REMOTE_ADDR	IP address of the remote host.
REQUEST_METHOD	The request method used, *e.g.* `GET`, `HEAD`, `POST` etc.
PATH_INFO	If present, the additional path information added at the end of the script URL.
PATH_TRANSLATED	The contents of `PATH_INFO` translated into a physical path. That is, the path to the resource on the server disk.
QUERY_STRING	If present, the query string following the script URL, *e.g.* everything after the "?" in the URL.
CONTENT_TYPE	For `POST` requests, the MIME type of the attached information.
CONTENT_LENGTH	For `POST` requests, the length of the attached information.
HTTP_USER_AGENT	Name and version number of the browser
HTTP_REFERER	The URL of the document that contains the link to the script URL. This is the document that sent the user to the script.
HTTP_COOKIE	A *magic cookie* passed to the server from the browser via the `Cookie` header.

This is only a partial list of the environment variables defined by the CGI protocol. Some servers extend the list of variables by adding extra variables that are not defined by the CGI protocol. In the case of the Apache server the extra variables defined can also depend on compile time options. You must always read the documentation of the server to find out exactly what variables are available.

Example 10.16: The information about the current request is stored in environment variables. If the current request was (without the break in the URL)

```
http://hostname:port/
            cgi-bin/formatpoem/coleridge/christabel.txt
```

then the following environment variables would have the values

`SCRIPT_NAME` = "`/cgi-bin/formatpoem`",
`PATH_INFO` = "`/coleridge/christabel.txt`",
`PATH_TRANSLATED` = "`/usr/local/web/coleridge/christabel.txt`"
Where we have assumed that the document root is `/usr/local/web`. The document root is the root directory where the server expects to find all documents.

10.5.2 The GET method

The GET method is the default method employed by the browser to request a document. It is also the default method for invoking a script. One of the environment parameters passed to the script is the `REQUEST_METHOD` which contains the value of the method used to invoke the script.

Keyword List

The keyword list (§10.3) is the original method used to send data to a script. It is the way the data entered in the `ISINDEX` HTML element is sent to the script.

The environment variable `QUERY_STRING` will contain the keyword list string. The string will be a "`+`" separated list of key phrases that are in URL-encoded form (that is characters not allowed in URLs will be encoded).

The keyword list will also be sent to the command line of the script where each phrase will be an element in the command line array (`char *argv[]` in C). The phrases will be unescaped, with one phrase per element of the command line array.

Which version of the list, either the parsed command line version or the raw `QUERY_STRING` version the script chooses to use is entirely up to the script author.

The amount of data that can passed to the script by the `GET` method is limited by the operating system the server is running on. The size of command line buffers and environment buffers is normally around a kilobyte.

Path Information

The path information (§10.3.2) is passed to the script in the two environment variables `PATH_INFO` and `PATH_TRANSLATED`. The path information is parsed so that URL escape codes are removed.

The `PATH_INFO` environment variable contains the parsed original string. The `PATH_TRANSLATED` environment variable contains the physical path to the resource on the host machine. This means that the document

root defined in the server configuration file (see the Server Configuration module 11) has been added to the `PATH_INFO` to make an absolute path.

Named Parameter List

The named parameter list (§10.10), like the keyword list, appears as the value of the environment variable `QUERY_STRING`. The string is not parsed and remains in its URL-escaped version.

Exercise 10.17: Use the script "`http.php`" and the examples above to check where and in what form the path information and the different lists are passed to the script.

Make certain to use URL encoding in conjunction with the special characters '`+`', '`?`' and '`&`' in the strings. It is important to note which are sent to the script URL encoded and which are not.

10.5.3 The POST method

Only the `Forms` tag allow the default method GET to be overridden using the `METHOD` attribute. The effect of the `POST` method is that the `REQUEST_METHOD` is set to `POST` and the `QUERY_STRING` will be *empty*. Instead of placing the query string in an environment variable, with the `POST` method the query string will have to be read by the script from *standard input*. There is no guarantee that the data on standard input will be line oriented, in fact it shouldn't as new line characters in the original query string will be URL escaped. There will be no end-of-string or end-of-file marker either. To read the data, scripts examine the environment variable `CONTENT_LENGTH` which contains the exact number of bytes to be read on the standard input.

The string read in on standard input will be URL-encoded and will have to be parsed.

Additional path information is handled the same for both `POST` and `GET` methods.

The `POST` method does not suffer from the limitations of the `GET` method — small buffers. This method is used when large amounts of data needs to be sent to the script.

Exercise 10.18: Use the script "`http.php`" and a simple `Forms` document to experiment with both the `GET` and `POST` methods.

Again note where the data appears to the script and whether it is URL-encoded or not.

10.6 Debugging Scripts

It can be tricky to debug scripts as they are under the control of the server not the author. There are however a few things that can be done to minimise headaches.

For PHP scripts run from the Apache PHP module:

- make use of the "`print`" and "`var_dump`" statements to output error information. Format all error messages in standard HTML so that the web browser can display the error message.
- Redirect error messages to the server's error log file using the "`error_log()`" function.
- Check the server error log file for error messages from the server or the PHP module.

For scripts written in other languages and that are not run by an Apache module:

- Test your program from the command line first. Make use of command line keyword query strings to test as much of your code as possible.
- Redirect *standard error* to *standard out*, or only send error messages to standard out. Format all error messages in standard HTML so that the web browser can display the error message.
- Messages sent by the script to *standard error* get redirected by the server into its error log file. Check the server error log file for script error messages. Be sure to output the script's error messages with the script name, as the log file can be large and difficult to read[4].
- Check the server error log file for error messages from the server. For example if you see the error message `Malformed header from script` you have probably forgotten to print the HTTP `Content-type` header line. Remember the header **must** be followed by an additional blank line before starting the text of the document.
- If the browser does something really weird when the script is run, such as wanting to place the output in a file and not display it, check that the MIME type specified with the `Content-type` header is correct. Remember it is this header information that tells the browser how to handle the data that follows.

10.7 Saving State Information

One of the limitations of the HTTP and the CGI interface is that it doesn't provide an easy way to keep track of a user's previous invocations of a script. Each time a user invokes a script, it's as if it were for the first time. This is a major drawback for scripts that need to maintain a long-running transaction. A couple of examples of scripts that need to maintain *state information* (that is the current *state* of the script, for example, the values stored in variables) are:

- Online shopping. A *shopping cart* script in which a user while browsing a catalogue adds to a growing list of purchases.

[4] The Apache log files can be found in the directory `~/CSC2406/logs`. The content format of the log files and where they can be found are all configurable.

- Multipart test or questionnaire, where questions on a page depend on the answers given on previous pages.

There are multiple ways to store state information, below are some, the choice of which method to use is dependent on the application, the preferences of the script author, and the security required.

10.7.1 Within Fill-out Forms

An obvious way to save state information is in the fill-out forms generated by the script. The first time the script is called the query string is empty and the script can use its default values. On subsequent invocations the query string contains values that the user submitted. Fields are repeated on subsequent forms with initial values submitted by the user on previous forms. Every time the script is called, it regenerates a new form based on the values of the old form, so the form's settings are preserved.

Sometimes though you do not wish the saved values to be displayed by the browser, either because the form would become too cluttered and difficult to follow, or you want to pass parameters back to the script without it being obvious in the document. In these cases the input field of type "`HIDDEN`" can be used. Any information you place in `HIDDEN` fields will be passed to the script in the query string but will not be displayed by the browser. You can place as many hidden fields in a document as you wish, containing any information you please.

Note

Do **not** use `HIDDEN` fields for important or sensitive information because they are not protected from tampering by the user. The user can view the source of any document passed to the browser. Any hidden fields can be modified by the user at any time.

10.7.2 Within URLs

If a script does not use fill-out forms, state information can be preserved directly in the script's URL.

When the script constructs a page, any links on the page pointing back to the script (or any other script) can have a query string added to the URL. Remember the query string (everything after the "?") can be in any style (as long as it is URL encoded). The keyword list and name/value pairs styles are just the two defined by the CGI/1.1 protocol. Anything in the query string will be placed in the environment variable `QUERY_STRING`.

So a script that produces a page that has links in it can add query strings to the URLs on the page and preserve state information across pages.

10.7.3 Within Path Information

Like the query string, the appended *additional path* information at the end of a URL can contain state information. One technique for maintaining state information is to write the information to a database or file using an encoded session ID. The session ID can be incorporated

into the *additional path* information of the URL and point to a file containing the state information of the session.

To ensure that the session id is always incorporated when the script is invoked the browser can be tricked into appending the path information every time the script is invoked. For example

(a) Script is invoked without a session ID in the additional path information of the URL.

(b) Script creates a session ID and the corresponding session file.

(c) Script sends a redirection request to the browser using the `Location` header field. The redirect to the browser reinvokes the script but with the session ID appended.

(d) Now every time the browser is requested to get the script *without* the session ID appended it will use the redirect URL which has the current session ID appended. The browser remembers a redirection URL for an entire session.

All links to the script in static documents will not have any additional path information making it easy to create them. The browser itself will add the session information to the URL after the first redirection request. This information of course will last while the browser remains running.

10.7.4 Using Authentication

Basic HTTP authentication requires the user to enter a username and password (see Module 12 for a fuller discussion on authentication). If your script requires a user to login to access it then the CGI protocol automatically creates a unique session ID for you, the user's *login name*. This can be used as a key into a database or disk file used to keep track of the session information.

When authentication is in effect, the script can find the user's login name in the environment variable `REMOTE_USER`, and the authentication type (usually the word "Basic" if username/password authentication is used) in the variable `AUTH_TYPE`.

10.7.5 Using Cookies

The most powerful and versatile way of retaining state information is with *cookies*[5]. Cookies are *name=value* pairs very much like the names parameters in the CGI query string. Unlike the query string, however, cookies are sent back and forth in the HTTP header rather than within the HTML URLs or forms.

Cookies have a number of important advantages over other methods for storing state information.

- Cookies are maintained by the browser, minimising the work of the script or server.

[5] The etymology of the term "cookie" is debatable.

- They can be associated with the entire site or a particular URL path. This allows a site to maintain a series of interacting scripts that share or pass state information to each other via cookies.
- Cookies can be assigned an expiration date. By default a cookie's lifetime is limited to the current session, but cookies can be created that will persist for days or longer.

There are also problems with cookies:

- The user (through the browser options) can determine whether cookies are accepted and stored. Scripts must therefore always allow for stateless interactions.
- Cookies and document caching can occasionally interact in strange and unpredictable ways. This is normally a problem with the proxy server not handling the non-caching properties of the cookies correctly.

Cookies are set and retrieved through a HTTP header. To create them, add one or more *Set-cookie:* fields to the header lines.

The syntax for the *Set-cookie:* header is (without the line break)

```
Set-cookie: CookieNAME=CookieVALUE; EXPIRES=date;
            PATH=path; DOMAIN=domain_name; SECURE
```

Example 10.19: The following code sets a cookie with the name *poet* and the value *Samual+Taylor+Coleridge*

```
Set-cookie:  poet=Samual+Taylor+Coleridge
```

In detail, the cookie options are:

cookieNAME=cookieVALUE A string of characters (excluding white space, commas and semicolons) specifying the *name* of the cookie and the value of the cookie. The value of the cookie is *opaque*, meaning it is not interpreted in any way whatsoever. The selection of the name and value pair are entirely up-to the script.

This is the only required option on the Set-Cookie header.

EXPIRES=date This attribute specifies a date string that defines the valid life of that cookie. Once the expiration date has been reached, the cookie will no longer be stored or given out.

The date string is formatted as:

```
Weekday, DD-Mon-YYYY HH:MM:SS GMT
```

The only legal time zone is GMT and the separators between the elements of the date must be dashes.

An example of a valid expiration attribute is

```
EXPIRES=Wednesday, 09-Nov-01 23:12:40 GMT
```

This attribute is optional. If not specified, the cookie will expire when the user's session ends. That is, when the browser is terminated.

DOMAIN=domain_name When searching the cookie list for valid cookies, a comparison of the domain attribute of the cookie is made with the Internet domain name of the host from which the URL will be fetched. If there is a tail match, then the cookie will go through **path matching** to see if it should be sent. "Tail matching" means that domain attribute is matched against the tail of the fully qualified domain name of the host. A domain attribute of "usq.edu.au" would match host names "www.sci.usq.edu.au" as well as "www.usq.edu.au".

The default value of domain is the host name of the server which sent the *Set-Cookie* header.

PATH=path The path attribute is used to specify the subset of URLs in a domain for which the cookie is valid. If a cookie has already passed **domain matching**, then the path name component of the URL is compared with the path attribute, and if there is a match, the cookie is considered valid and is sent along with the URL request.

The path "/x" would match both "/xy" and "/x/y.html". The path "/" is the most general path.

If the path is not specified, it as assumed to be the same path as the document being described by the header which contains the cookie.

SECURE If a cookie is marked secure, it will only be transmitted if the communications channel with the host is a secure one.

If secure is not specified, a cookie is considered safe to be sent in the clear over unsecured channels.

Cookie HTTP Request Header

When requesting a URL from an HTTP server, the browser will match the URL against all cookies and if any of them match, a line containing the name/value pairs of **all** matching cookies will be included in the HTTP request. The format of the cookie HTTP request header sent by the browser is

```
Cookie:  cookieNAME1=OPAQUE_STRING1; NAME2=OPAQUE_STRING2 ...
```

A server script can access the cookie request header through the environment variable `HTTP_COOKIE` where it has been copied by the server.

Example 10.20: The following code sets a cookie for the path `/poets`

```
Set-Cookie: poet=Samual%20Taylor%20Coleridge; path=/poets;
     expires=Wednesday,  09-Nov-01 23:12:40 GMT
```

A request for a document contained in the path `/poets` would generate a cookie request header by the browser of

```
Cookie: poet=Samual%20Taylor%20Coleridge
```

A subsequent *Set-Cookie* request of

```
Set-Cookie: poem=The%20Rime%20of%20the%20Ancient%20Mariner;
        path=/poets/poems
```

would mean that a request for a document in the path "`/poets/poems`" would generate a cookie request header by the browser of

```
Cookie: poet=Samual%20Taylor%20Coleridge;
        poem=The%20Rime%20of%20the%20Ancient%20Mariner
```

All cookies matching the *path attribute* are sent.

Exercise 10.21: Modify the server script "http.php" to send cookies when it receives a specific keyword, such as "cookie". For example

```
http://hostname:port/http.php?cookie
```

Should generate the following header

```
Set-cookie:  myCookie1=testValue1
```

Experiment with alternate paths for cookies. Modify the script to read the "extra path information" and add this path information to the set-cookie command. For example

```
http://hostname:port/http.php/alt/path?cookie
```

This command would generate the following header

```
Set-cookie:  myCookie2=testValue2; PATH=/alt/path
```

Copy "http.php" around your document tree and access it from different directories. See which cookies are sent from the browser. Give each cookie a different name and value so that they can be identified.

Exercise 10.22: What happens if you resend a cookie to the browser with different attributes and a different value but the same name?

How do you think a script can delete a cookie from the browser's list of stored cookies?

Exercise 10.23: PHP has the ability to specify "sessions"—that is, a consistent way to preserve certain data across subsequent accesses by a client.

Read the PHP manual section on sessions.

Modify the "http.php" script so that a number of variables are registered for the current session using the global variable `$_SESSIONS`. Output the contents of this variable each time the script is called.

Explain how "cookies" and `$_SESSIONS` variable are different.

10.8 Questions

Short Answer Questions

Q. 10.24: What is a *dynamic document*?

Q. 10.25: How does the client request the server to run a script?

Q. 10.26: How does the client parse information to a script?

Q. 10.27: What characters must be escaped before passing them to the server script?

Q. 10.28: How does the server know that a particular URL points to a script?

Q. 10.29: How is a keyword list sent to a script?

Q. 10.30: The `POST` and `GET` methods and the `ISINDEX` tag all send data to server scripts but in different ways. How are they different?

Q. 10.31: What are *name/value* pairs? How are they sent to a script?

Q. 10.32: How is path information sent to a script?

Q. 10.33: Can name/value pairs and path information be combined?

Q. 10.34: How is the content of a Forms page broken into name/value pairs?

Q. 10.35: Why do scripts have to add the `Content-Type` header to their output to the client.

Q. 10.36: How do scripts send data to the client?

Q. 10.37: What is a *redirection directive*?

Q. 10.38: What is the *Common Gateway Interface*?

Q. 10.39: Where does the CGI define where the `POST` method data is to be found? The `GET` method data? The `ISINDEX` keywords?

Q. 10.40: What sort of information is passed via environment variables?

Q. 10.41: What information is passed to the script as command line arguments?

Q. 10.42: The path information sent to the script can be found in the two environment variables `PATH_INFO` and `PATH_TRANSLATED`. What information is found in each variable?

Q. 10.43: Which information passed to the script is parsed so that URL escape codes are remove? Which information is not parsed?

Q. 10.44: When is the environment variable `CONTENT_LENGTH` required by scripts?

Q. 10.45: What is *state information*?

Q. 10.46: Why are server scripts considered *stateless*?

Q. 10.47: How do server scripts maintain state information? List all possible ways.

Q. 10.48: How are cookies set by a script? How are they retrieved?

10.9 Further Reading and References

(a) Browse the PHP manual available on the course web site.

(b) The cookie specification was first proposed by Netscape Communication Corp. The original white paper can be found at their web site.

(c) An alternate cookie specification, that is slightly incompatible with the Netscape specification, has been proposed in RFC 2109 and RFC 2965.

Chapter 11 Server Configuration

This module covers *some* of the aspects of configuring a server. It does not cover *all* of the available configuration directive available to the web administrator. Apache 2.x has approximately 350 directives. All of the Apache directives can be found in the Apache documentation under the link "Run-time configuration directives".

The Apache server is written in a modular form. Features are associated with modules, add a module and add that module's features—also add that modules configuration directives. Normally, only a subset of modules are available, which means only a subset of configuration directives are available.

This module will concentrate on *core* features of servers, features that all servers must implement in some form or another to conform to the HTTP. You will not be expected to memorise all directives discussed in this module but you must be familiar with the major directives that change the behaviour of the server.

Chapter contents

11.1 Introduction

There are a myriad of servers available today, both freeware and commercial. All have their advocates, but apart from differences due to operating system idiosyncrasies and cosmetic administration features (such as a GUI) all are based on the original servers developed at

CERN and NCSA. Also, servers must be able to understand the HTTP and the CGI which restricts the addition of non-standard features.

One of the most popular servers is the Apache Server. A freeware server that grew directly from the original NCSA server and therefore inherited that server's features and structure. Though originally written for Unix operating systems, Apache is available for Microsoft systems as well.

11.2 Global Configuration Files

All servers will have some variation on the configuration file or files. Even servers with an administration GUI will have configuration files, which can be either changed by hand or through the GUI.

The Apache server has at least two configuration files that effect the global operation of the server, they are `httpd.conf` and `mime.types`

The file `httpd.conf` contains the basic operating parameters, document and server tree definitions, access control *&c.* of the site and is read by the server at start up. This file is normally split into a number of independent files for ease of administration. At the end of the `httpd.conf` file will be `Include` directives that define additional configuration files that should be loaded at start up.

The file `mime.types` contains the list of MIME types that the server will recognise. This file is normally complete and never need be touched. The form of the file is straight forward with the MIME type defined first followed by the list of file extensions. The server maps the file extension to the MIME type and adds the MIME type to the document through the `Content-Type` header field[1]

All configuration files follow the same format, a `#` symbol starts a comment line, non-comment lines start with the configuration directive followed (on the same line after white space) by the configuration value.

Exercise 11.1: Study the Apache configuration files, which can be found in the directory `~/CSC2406_Server/conf/` and its subdirectory `extra`.

The configuration directive `Include` is used to include additional configuration files. The main configuration file is split for ease of maintenance only. It allows directives to be grouped by module and purpose.

Read the comments (lines beginning with the hash/sharp/pound symbol `#`) associated with each configuration directive.

Apache's default configuration file does not contain all the possible configuration directives, it only contain the most frequently used, or the directives that need a default value defined. The server documentation contains a complete list of the server configuration directives.

[1] Apache can also be compiled with a `MIME magic` module. This is a module that will read the first few bytes of a file and try and ascertain the MIME type of the file by looking for *magic bytes* that identify the file and any associated application.

Exercise 11.2: The course web site has a link to the documentation of your version of Apache. Using the descriptions of key configuration directives below as a reference, browse the server documentation and study the complete list of configuration directives.

Configuration directives come in two flavours, *Global* and/or *Local*, where local means that they can be applied at the directory level of the server's document tree. The directory configuration directives apply only to the directory specified, *and its sub-directories*. If allowed, these sub-directories can have there own configuration directives. This creates an hierarchy of configuration directives. If allowed, later directives (in the hierarchy) override earlier directives. This feature allows the different parts of a Web site to be configured very differently to each other.

We will look at the global directives first then see how to achieve finer control over the document tree with directives for individual directories.

11.3 Global Configuration Directives

The following define some of the Global configuration directives. These directives are only defined for the global configuration file that Apache runs at start up. These directives then apply to the entire site.

11.3.1 The Root Directories

DocumentRoot directive

The documents that a server are to make available to the world are stored in the *Document Tree*. The tree is a hierarchy of directories containing the pages of the site.

The *root* directory of the document tree is called the *document root*—the *document root* corresponds to the top level URL for the site, that is the URL path '/' represents the document root.

Example 11.3: If the server is configured with the document root as

`/usr/local/www`,

then the URL—

- `http://hostname/`

  will direct the server to look for the document

  `/usr/local/www/`.

  The trailing slash of the above URL means "the document root".

- `http://hostname/poets/colleridge.html`

  will direct the server to look for the document

```
/usr/local/www/poets/colleridge.html
```

At the top level of the document tree, the document root, will be the *welcome page* for the site. The *welcome page* is the document referenced by the site URL `http://hostname/` (See §11.4.3 on how to modify this behaviour).

ServerRoot directive

The *server root* directory is where the server software and all its support files are stored. Among the files found here are log files, configuration files, icons used internally by the server, maintenance utilities *&c.* More importantly, it is here that the directory containing the CGI scripts is normally placed.

The server and document roots may be located together or kept separate. A common practice (and the default setting for Apache) is to place the document root inside the server root in order to keep all web-related files together.

Listen directive

This directive defines the port the server will listen on. This is normally port 80. If the port number is changed from the expected port of 80, then the URL of the site must incorporate the port number defined by this directive.

User and Group directives

When any process is run on the Unix system it belongs to a particular user and group. The user and group a process runs under defines the privileges it has. By specifying a restricted user and group that the server will run under can limit the damage done by buggy server scripts run by the server. Like any user—scripts started by the server are owned by the server. The standard user and group for a server on a Linux system is `nobody` (the least privileged of all users). For example:

```
User  nobody
Group nobody
```

11.3.2 Virtual Document trees

Alias directive

A single document root is too restrictive in practice, it is difficult to maintain a document tree with one root. A useful feature provided by Apache, and many other servers is the ability to support *virtual* document trees. This allows the administrator to combine different physical directories into a unified single hierarchy of URLs.

Example 11.4: By using the Apache `Alias` directive

```
Alias /photos/ /mnt/cdrom/
```

you can arrange for the physical directory `/mnt/cdrom/`, a mounted CDROM, to become part of the document tree. The server now maps URLs beginning with

`http://hostname/photos/`

to the physical location `/mnt/cdrom/`—a location independent of the document root of the site.

The virtual document tree feature allows a site to spread out across additional disks (including disks mounted across networks) as it grows—but still maintain the same URL hierarchy.

ScriptAlias directive

A special form of the virtual document tree is the directory (or directories) containing CGI scripts. This directory is normally part of the server root, not the document root. The CGI script directory is kept separate from the document directories as it will have special privileges reflecting the special status of the files it contains. Also for administration purposes it is convenient to keep executable scripts in one place.

The `ScriptAlias` directives dual purpose is to map an external directory into the document tree, exactly the same way as the `Alias` directive, but it also tells the server that the aliased directory will contain *only* executable scripts. Any resource requested from the script directory will be executed by the server, not returned to the web browser.

There is no limit on the number of `Alias` or `ScriptAlias` directives that can be defined.

Note

This is only required for scripts or executables that are run by the Web server and are independent of it. Since the Apache Web server has a PHP module—PHP scripts are interpreted by the server and not executed independently, therefore they do not need to reside in a special directory

11.3.3 User Directories

A special form of the virtual document tree is *user supported directories* in which portions of local users' home directories are made part of the document tree. Access to user supported directories are through URLs starting with the tilde "~" character. When the server recognises the tilde it maps the username following the tilde to the user's personal web directory[2].

UserDir directive

Apache uses the directive `UserDir` to specify the *name* of the directory, in the user's home directory, that will contain the personal web pages.

[2] The server's interpretation of a tilde derives from the Unix shell directive "~" which signifies the user's home directory, and the "`~user`" directive which signifies the home directory of *user*.

Example 11.5: The Apache default directory for personal web pages is `public_html`. This means that a URL of the form

`http://hostname/~w0012345/`

will be mapped to the directory `public_html` in the home directory of user `w0012345`.

If the home directory is `/home/student/w0012345/` then the URL maps to `/home/student/w0012345/public_html/`

A variety of syntaxes are allowed, giving rise to interesting variations. For example if a user requests the document

`http://hostname/~w0012345/results/CSC2406.html`

then the actual resource retrieved depends on the form of the directive:

Directive: `UserDir public_html`
File Retrieved: —
`/home/student/w0012345/public_html/results/CSC2406.html`

Directive: `UserDir /usr/web`
File Retrieved: `/usr/web/w0012345/results/CSC2406.html`

Directive: `UserDir /home/*/www`
File Retrieved: `/home/w0012345/www/results/CSC2406.html`

Directive: `UserDir http://newhost/*`
File Retrieved: — `http://newhost/w0012345/results/CSC2406.html`

See the Apache manual for all possible variations on the syntax for `UserDir` mappings.

11.3.4 AccessFileName directive

The directory access control file is a file that can be placed in any of the directories found in the document tree. This file contains configuration options for the server. The options found in this file, unlike the global options, only apply to the directory containing the access file. The options also apply to any of the directory's sub-directories.

If the file does not exist in a directory then access to the directory is dependent on the global configuration file, or the access control files in parent directories.

This directive sets the name of the directory access file. The default is *.htaccess*. For this reason the directory access file is commonly called the "htaccess file".

11.3.5 <Directory . . . > directive

The directives in the configuration file adjust global settings for all the directories at a site. To achieve finer control over the document tree, options can also be set for individual directories, either by changing the central configuration file or by placing a configuration file in the directories themselves.

The global configuration file for the site is normally controlled by the site's administrator and it sets global configuration policies for the whole site. On the other hand, individual sections of a large site may be controlled by that section's maintainer. A large site can have many site users only controlling sections of the site. By enabling directives in the global configuration file that allow control of individual directories the site's maintainer can place limitations on specific sections of the site, restricting the power of that section's maintainer (Consider the danger that User Directories pose to a Web administrator).

In the global configuration file only, to apply directives to a specific directory tree in the site the directives must be grouped so they only apply to a specific directory. Configuration directives set for a specific directory are automatically inherited by all its sub-directories. If a sub-directory has its own configuration section, its directives override its parent's configuration — options are not additive.

To specify the directory to be modified either the full path name of the directory can be used (the `<Directory...>` directive), or the URL of the directory can be used (the `<Location...>` directive). Wild card characters can be used when specifying directories so that the directives apply to all directories that match the directory specification.

To group directives so that they apply only to a specific *physical* location in the document tree then the directives must be bracketed by the begin directory — end directory pair, `<Directory ...>` and `</Directory>`.

To specify the directory, the full physical path name of the directory must be specified. All directives placed between the begin directory — end directory pair *only* apply to the specified directory and all its sub-directories.

Example 11.6: Suppose the document root of the site is `/home/www/` then to modify the directory `/home/www/poets/colleridge`, the modifying directives must be within the following `Directory` directives

```
<Directory /home/www/poets/colleridge>
...
...
</Directory>
```

If the directory `/home/www/poets/colleridge` contains a directory called `poems` then the directives defined for the parent directory `colleridge` also apply to the child directory `poems`.

By placing directory configuration directives in the global configuration file the site administrator can control how and to what extent users of the site can modify the server's configuration in their own sections.

11.3.6 <Location ...> directive

The `<Location ...>` directive is the same as the `<Directory ...>` directive except while the `<Directory ...>` directive takes the file

system name of a directory as a parameter the `<Location ...>` directive takes the URL of the directory. For example,

```
DocumentRoot /usr/local/www

<Directory /usr/local/www>
  Options Indexes FollowSymLinks MultiViews
</Directory>

<Location />
    order allow,deny
    allow from all
    deny from .au
</Location>
```

because the document root is `/usr/local/www` the `<Location ...>` directive and the `<Directory ...>` directive refer to the same directory.

Example 11.7: Using Example 11.6, the same directory can be specified as

```
<Location /poets/colleridge/>
...
... (Place directives here)
...
</Location>
```

The document root is `/home/www`, so the URL maps to

`/home/www/poets/colleridge`.

Which is the same directory as the previous example.

The advantage of using the `<Location ...>` directive to specify a directory is that URL paths are generally shorter and are less likely to change if the physical layout of the site is modified.

Example 11.8: Wild card characters can be used when specifying directories.

Suppose the user directories are all stored under the directory `/home`, then one way to turn off all options for user directories is

```
<Directory /home/*/public_html>
Options None
</Directory>
```

or

```
<Location ~*>
Options None
</Location>
```

11.3.7 AllowOverride directive

This directive determines whether the configuration directives specified for the current directory (and by default its sub-directories) can be overridden using access control files (see §11.4).

When the server finds an access control file in a directory it needs to know which directives declared in that file can override earlier access information.

Note This directive is extremely important one for the Web administrator. This is the directive the Web administrator can use in the global configuration file to restrict the configuration directives available to the individual section maintainers of the site.

The allowed values for the directive are

`All` The directory access file can override all earlier directives. This is the default

`None` The directory access file cannot override earlier directives. In this case the directory access file in the directory is not even read.

`AuthConfig` The directory access file can override authorisation directives, such as `AuthGroupFile`, `AuthName`, `AuthType`, `AuthUserFile`, `require` *&c.* (see module 12 on Security).

`FileInfo` The directory access file can override directives controlling document types, such as `AddEncoding`, `AddLanguage`, `AddType`, `DefaultType`, `ErrorDocument`, *&c.* (See sections below)

`Indexes` The directory access file can override directives controlling directory listings, such as `DirectoryIndex`, `FancyIndexing`, `AddDescription`, `AddIcon`, *&c.* (See §11.4.3 below)

`Limit` The directory access file can override directives controlling host access. (see module 12 on Security).

`Options` The directory access file can override directives controlling specific directory features, such as the `Options` directive. (See §11.4.1 below)

Exercise 11.9: Suggest which options above should not be allowed for user directories.

11.4 Directory Access Control Files

An alternative to specifying per-directory options in the global configuration file, is to place an *access control file* at the top of every directory tree you wish to modify. The access control file is a plain text file with the name specified by the `AccessFileName` directive (the default name is *.htaccess*). The access control file contains the directives that can also be placed between the directory grouping directives in the global configuration file.

Note

An important point to remember about access control files, though they should be writable by trusted users only, they **must** be readable by the *server*, which is in effect the *world*.

Exercise 11.10: Read the documentation on the Apache run-time configuration directives. These can be found either at the course site, or part of your own Apache installation.

Each directive specifies the *Context* in which the directive can be used. Of interest is *server config*, *directory*, and *.htaccess*.

- *Server config* implies the directive is part of the server configuration and can only be found in the global configuration files.
- *Directory* implies that the directive is part of a directory configuration group and must be placed within the

  `<Directory ...> ... </Directory>`

  or

  `<Location ..> ... </Location>`

  directives.
- `.htaccess` implies the directive can be placed in a directory access file.

Some directives can be placed in all three locations, some can't. Study the directives (there are a lot more of them than will ever be discussed in this study book)[3]

Note the *type* of directives that can be placed in the various locations, especially the type of directives that can be used in a *.htaccess* file.

Note

The `Directory` and `Location` directives themselves most definitely *cannot* be placed in a directory access file. The two directives are used in the global configuration file to define which directory/location to apply the grouped directives.

The directory access file applies its directives to the directory it resides in and does not need or want a grouping declaration. The server will return a 500 status code "Internal Server Error"

11.4.1 Options directive

This directive controls the basic features of the directory. Each option is a trade-off between convenience and security. For example, allowing the server to follow symbolic links opens up the possibility of someone (either deliberately or accidentally) creating a link from the public web area to a more private part of the system.

The available options are:

[3] Your are **not** expected to memorise the names of all the Apache directives. You are expected to know the issues discussed in the study book and that there are groups of directives designed to perform specific tasks.

`None`	No features are enabled for this directory
`All`	All features are enabled for this directory. Except the `MultiViews` parameter, that must be set explicitly. This option is the default directory setting.
`FollowSymLinks`	The server will follow symbolic links in this directory, wherever they lead!
`SymLinksIfOwnerMatch`	The server will follow symbolic links only if the link target is owned by the same user as the link itself.
`ExecCGI`	Executable scripts (CGI scripts) are allowed in this directory.
`Includes`	Server-side includes are allowed in this directory.
`IncludesNoExec`	Server-side includes are allowed in this directory, **but** the `exec` and `include` commands are turned off.
`Indexes`	Automatic directory listing is allowed for this directory if a *welcome page* has not been defined or is not present.
`MultiViews`	Turns on content negotiation for this directory

11.4.2 Redirection

Occasionally a file or an entire directory on a server needs to be moved. This could happen because the site has outgrown the hardware and the directory structure needs to be reorganised. In this case a directive can be used to redirect the browser (using the 3xx status codes, see Table 7.3) to the new location of the document.

Redirect Directive

This directive is similar to the `Alias` directive but the second parameter (the new location of the document) can be a complete URL not only a physical location.

Example 11.11: Suppose the directory `/poets/colleridge/` had become so large and generating so much activity on the site that it was decided to move it to its own site

`http://poets.org.au/colleridge/`,

then the directive (which should be all on one line)

```
Redirect permanent /poets/colleridge/
    http://poets.org.au/colleridge/
```

would generate a 301 status code response from the server, with the `Location` header field containing the new address for the document. The web client would then silently re-request the document using the new URL.

The `Redirect` directive has an optional status field that specifies the status code response for the server. The values of the optional status parameter are `permanent` (301), `temp` (302), `seeother` (303), and `gone` (410), the default status code is 302.

11.4.3 Directory Resources

When a web browser requests a URL that points to a directory resource rather than a file resource the server must decide how to handle the request. It obviously cannot return the contents of the directory. There are two options defined for web servers in this situation, they can either synthesis an HTML file that lists all the files in the directory, or they can search for a default file in the directory to display.

DirectoryIndex directive

This directive sets the name of the *welcome page* that the server will attempt to display when a browser requests a URL that ends in a directory name rather than a file name. When the server receives a request for a URL that points to a directory the server looks inside the directory for a file matching one of the file names specified by the `DirectoryIndex` directive. If a file is found that matches one of the file names in the list this file is returned. If no matching file is found and automatic directory listing is turned on the server will construct a list of all the files in the directory, in HTML, and send that back to the browser. If no matching file is found and if automatic directory listing is turned off then an error message is returned to the browser.

By default the welcome file is called *index.html*, the idea being that this file is an index of the contents of the directory.

Example 11.12: The directive

```
DirectoryIndex index.html index.php index.cgi
```

tells the server to look, *in order* for the files `index.html`, `index.php`, and `index.cgi`. The first is a static document, the second a PHP script and the last a CGI executable. If the file is found it is returned, if the PHP script is found it is parsed, and if the CGI script is found it is executed and its output is returned.

Automatic Directory Synthesis

Most servers have the ability to synthesise a directory listing on the fly when a request is received for a directory lacking a welcome page file (defined by the `DirectoryIndex` directive). For security reasons automatic directory listings should be turned off and welcome pages supplied for all directories (this will be discussed in greater detail in module 12 on security), but situations arise where automatic directory listings are a useful feature, for example a directory where the contents are changing rapidly because multiple people are using it as a "drop box", when maintaining a directory tree shared by both FTP and Web servers.

To avoid having to supply a welcome page with links to every file in the directory, the server can synthesise an HTML page containing links to every file in the directory.

Automatics directory listings can be turned on for a directory using the `Options` directive. For example the directive

`Options Indexes`

turns on automatic directory listing for the directory tree it appears in.

Apache supports two styles of directory listing a compact *simple* style and a more elaborate *fancy* style.

Example 11.13: A *simple* directory listing

- Parent Directory
- audio/
- commentary.mp3
- image1.png
- image2.jpg
- lecture01_2p.pdf
- lecture01_4p.pdf
- lecture01_s.pdf
- video.mp4

A *fancy* directory listing

Name	Last modified	Size	Description
Parent Directory		-	
audio/	06-Jul-2011 13:53	-	
commentary.mp3	06-Jul-2011 15:20	11K	Lecture Commentary
image1.png	06-Jul-2011 15:19	2.5K	Lecture Images
image2.jpg	06-Jul-2011 15:19	7.5K	Lecture Images
lecture01_2p.pdf	06-Jul-2011 13:54	7.5K	Lecture Slides
lecture01_4p.pdf	06-Jul-2011 13:54	5.0K	Lecture Slides
lecture01_s.pdf	06-Jul-2011 13:53	12K	Lecture Slides
video.mp4	06-Jul-2011 15:20	11K	Lecture Video

This text is displayed because it is in the file **README.html** and is added at the bottom of the listing.

The compact *simple* style lists the files in the directory by name, each name a link to that file. The *fancy* indexing style adds file size, modification date, MIME type information in the form of an icon, as well as an optional short description of the file.

The fancy indexing icons are defined, by MIME type, by file extension, or by Content-Encoding. If the server can not resolve the icon for a file then a default icon is employed.

Some of the directives that control directory listing are:

`FancyIndexing` Turn on fancy indexing.

`AddIcon` Set the icon to display next to a file. The file name is defined by its type extensions. For example,

```
AddIcon /icons/image.xbm .gif .jpg .xbm
```

`DefaultIcon` The icon image to display for a file when no icon image is known.

`ReadmeName` Set the name of the file to display at the end of the directory listing.

`HeaderName` Set the name of the file to display at the top of the directory listing.

`AddDescription` Define the descriptive string to add to a file's listing. For example,

```
AddDescription "The planet Mars" /web/pics/mars.png
```

`IndexIgnore` Add to the list of files to *ignore* when creating the directory listing. Wild-card expressions can be used for the file names.

`AddIconByType` Set the icon to display next to a file. The file name is defined by its MIME type. For example,

```
AddIconByType /icons/image.xbm image/*
```

`AddIconByEncoding` Set the icon to display next to a file. The file name is defined by its encoding. That is, by the compression method used to compress the file. The compression method is defined by its MIME encoding. For example,

```
AddIconByEncoding /icons/compress.xbm x-compress
```

11.4.4 ErrorDocument Directive

The normally uninformative error messages returned by the server and displayed by the browser (for example, the infamous error message *Status 404, Document not found*) can be replaced using the `ErrorDocument` directive. If a document is not found, a site search page could be displayed, or if there is an authorisation failure when a user attempts to access a restricted document then instructions could be displayed on how to subscribe to the site and gain access to restricted documents.

Exercise 11.14: An example of a local `404, Document not Found` response is the site

`http://www.sci.usq.edu.au/`.

Request an nonexistent document from this site.

The directive takes two parameters:

(a) the HTTP status code for the error, and

(b) the URL of the document to display, or a string to display in well formatted HTML.

For example:

```
ErrorDocument 404 /status/notFound.html
```

The document to be displayed can be local to your site, located elsewhere, a static document, a PHP script, or a CGI executable.

Exercise 11.15: Create a page to be displayed when a 404 error occurs. Install it on your server and modify the appropriate configuration file so that the server will display your page instead of the default message.

Test your page and configuration by requesting a nonexistent document.

11.4.5 Encodings and Languages

Apache supports content negotiations. This feature allows the server to choose from several alternatives, the file that is most preferred by the browser requesting it. The alternative files can have language differences, MIME type differences or encoding differences (different compression methods).

Server content negotiation is specified on a directory by directory basis in Apache by setting the option `MultiViews` (see §11.4.1 above).

AddType Directive

As outlined in Module 8, Web servers use the MIME typing system to tell browsers what kind of document the browser is receiving. Browsers need this information so they know how to display the document. The main way Apache decides on a document's MIME type is to match the document's extension against the list in the *mime.types* configuration file. To add MIME types it is a simple matter to edit this file. However, this file contains the official list of MIME types recognised world-wide. It is generally better to leave this list alone, and use the configuration directive *AddType*, to add a new, local MIME type to the server list.

The *AddType* directive takes two parameters, the new MIME type and the file extension. To attach more than one extension to a MIME type use multiple *AddType* directives (this is different to the *mime.types* configuration file).

Example 11.16: The declarations

```
AddType image/x-fauna  dog
AddType image/x-fauna  cat
AddType image/x-fauna  rat
```

would tell Apache to consider all files that ended in the extensions `dog`, `cat` and `rat` to be new image files of type *image/x-fauna*. The *AddType* directive takes precedence if the MIME types have already been defined in the *mime.types* configuration file[4].

AddEncoding directive

Compressing files can significantly speed up downloads across the network. The *AddEncoding* directive is used to support clients who can *unpack* compressed files on the fly. The effect of this directive is to

[4] Remember when adding your own MIME types to adhere to the convention of beginning experimental types and subtypes with an *x-*.

add the *Content-Encoding* field to the header of all the files ending with one of the indicated suffixes.

The advantage of this directive is that files can be compressed without loosing the underlying MIME type of the document. That is, if a document has multiple file types, for example `CSC2406.html.gz` then the *Content-Encoding* and *Content-Type* headers will be

```
Content-Encoding: gzip
Content-Type: text/html
```

The client will know the HTML document has been compressed.

DefaultType directive

This directive tells Apache what MIME type to use when it can't determine the type from the extension. The most common choices for the default MIME type is either *text/plain* or *application/octet-stream*. The first declares the document as a plain text document which the browser will try to display, the second as a binary document the browser will write to disk and let the user sort out what to do with it.

AddLanguage directive

Language negotiation means that the browser can negotiate with the server for the preferred language of the documents (if they are available). When language negotiation is active on the server (by setting the MultiViews option), multiple copies of the same document, in different languages, can be placed together in the same directory. When the browser requests a document with multiple language versions, the document most preferred by the browser can be returned.

The default configuration file has the *AddLanguage* directives for about 20 languages[5]

Example 11.17: Suppose you have a manual in three languages,

- `manual.html.xh` Xhosa version
- `manual.html.tl` Tagalog version
- `manual.html.sa` Sanskrit version

when the browser requests the document using the base name *manual.html*, the browser's most preferred language version of the document will be returned by the server.

[5] The current Language and Country codes for the Internet can be found in ISO 639-1 and ISO 3166 respectively. Language codes were originally defined as 2 letter codes (ISO 639-1) but the number of combinations of 2 letter codes is too small for the number of languages so the new codes are 3 letters. The resources directory contains pages listing both ISO 639-1 and ISO 3166

LanguagePriority directive When the browser does not support language negotiation the server does not know which language has priority so the directive *LanguagePriority* is used to define which language has priority. It is also used to break a tie. The languages are listed in decreasing order of priority.

Exercise 11.18: Create a new directory in your document tree and place in it multiple copies of the same document in different languages. (if you know only one language — *fake it!*)

Add a directory access file to the directory and

- turn on content negotiation
- add your language extensions
- prioritise your languages.

The server should now be able to identify your language extensions and be able to negotiate with a browser on the most preferred language.

From your browser request the generic document (without specifying the language extension). Which document do you expect to receive? Why?

Modify your browser to accept your language extensions. Modify the language order of preference. Does negotiation occur? Modify the preferences again? Do you get a different document?

11.4.6 Handlers

Apache offers a general handler mechanism to attach special actions to certain types of documents. This mechanism is used to implement CGI-scripts, clickable image-maps, server-side includes and other features.

Ordinarily, when the server is asked to retrieve a URL, it finds the file that the URL corresponds to, uses the suffix to determine the appropriate MIME type, and sends the contents back to the browser. However if a *handler* is defined for that file type, the file is passed to the handler software, which processes the file in some way and returns the results.

Handlers can be internal to the server, that is compiled into the server, or implemented externally by a CGI script.

The following are some of the internally implemented handlers defined by the Apache server;

`send-as-is`	Send the file as is without adding HTTP headers.
`cgi-script`	Treat the file as a CGI script.
`imap-file`	Imagemap rule file.
`server-parsed`	Parse for server-side includes
`type-map`	Parse as a type map file for content negotiation

(See the Apache documentation on handlers for a complete list of the server handlers).

The following are some of the directives used to assign handlers:

`AddHandler` Associate a handler with a file suffix. For example, to tell Apache that the suffix `cgi` is associated with scripts to be run by the `cgi-script` handler, the directive is

```
AddHandler cgi-script cgi
```

`ScriptAlias` Alias a directory containing CGI scripts. For example

```
ScriptAlias /cgi-bin/ /home/web/cgi-bin/
```

`SetHandler` All files of the appropriate type in the directory will be parsed by the handler. This directive is used within a directory access file, a `<Directory>` directive or `<Location>` directive, and defines the handler for the directory's content.

`Action` Associate a CGI script with a MIME type. Every time a file with the `Action` MIME type is requested, the file is parsed by the associated CGI script. For example,

```
Action text/html footer.php
```

Every time a document of type `text/html` is requested, it is parsed by the script `footer.php`, which could add a standardised footer to every HTML page.

`Script` Associate a CGI script with an HTTP request method. For example:

```
Script PUT /cgi-bin/upload.php
```

When a `PUT` request is issued from a browser it is passed to the script `upload.php`.

11.4.7 Imap Files

To handle clickable image maps, Apache internally implements a handler called *imap-file*. It reads map files that define the hot regions in clickable image maps and directs the browser to the appropriate URL when the user clicks the mouse in one of those regions.

Example 11.19: To declare that *imap-file* is the handler for map files with the suffix *.map*, the following directive can be added to either a directory access file or the global run time configuration file

```
Addhandler imap-file map
```

Exercise 11.20: Study the Apache documentation on Imap files (see the `mod_imap` documentation). This documentation gives the necessary directives and examples of creating imap files.

Create a directory to hold your image, imap file and documents linked via the image map.

Create, or find on the Web, an image you like.

Create the image map file for your image specifying at least 3 hot spots

Test your map.

11.5 Questions

Short Answer Questions

Q. 11.21: What is the difference between the *Document Root* and the *Server Root*?

Q. 11.22: What is the *Welcome Page*?

Q. 11.23: What are *virtual* document trees? How are they implemented? Why would you want virtual document trees?

Q. 11.24: Why is it a good idea to keep CGI scripts in a separate directory?

Q. 11.25: How would you redirect the client to the correct URL when a document or document tree has moved? How does the server inform the client of the document or document tree's new location?

Q. 11.26: What is *automatic directory listing*?

Q. 11.27: How does the server know that the requested URL is a directory?

Q. 11.28: What is the *.htaccess* file? What is it's purpose? Why should it's name be changed from the default?

Q. 11.29: What is the purpose of the *AddType* directive?

Q. 11.30: How does the server implement *Language Negotiation*?

Q. 11.31: Explain the difference between *Global configuration directives* and *Directory configuration directives*

Q. 11.32: How do you specify the directory you wish your directives to apply to in a *Global configuration file*? In a *Directory configuration file*?

Q. 11.33: What is the difference between the *Location* and *Directory* directives? What are they used for?

Q. 11.34: What is the purpose of the *access* control file?

Q. 11.35: What are the purpose of *handlers*? What is the difference between *internal* and *external* handlers?

Q. 11.36: Explain why the `Directory` and `Location` directives cannot appear in a directory access file.

11.6 Further Reading and References

(a) The Apache configuration files supplied with the server. Remember these files only contain the basic configuration directives needed to set up a vanilla server.

(b) The Apache documentation supplied with the server. This documentation list the complete set of directives available for the server. Do not be daunted by the number of directives, you are not expected to remember individual directives by name. You are expected to remember the ideas and concepts that group directives.

(c) The Apache documentation has a page on "Content Negotiation" that outlines the method used. A link can be found, at the top level in the Apache manual.

(d) Apache documentation on Handlers. This explains how Apache assigns implicit or explicit handlers to parse file types. A link can be found, at the top level in the Apache Manual.

Chapter 12 Server Security

This module covers some of the aspects of web security. That is, maintaining the integrity of your site from malicious attack or simple stupidity. It will also cover some of the more advanced aspects of web security, user authentication and communication integrity.

Chapter contents

12.1 Introduction

The Web's power to open the site to the world also exposes the site to security risks. The type and degree of the risk varies from the well-meaning internal user (or web administrator for that matter) who unwittingly creates a symbolic link (shortcut) that opens up a private part of the system to public perusal, to the malicious hacker intent on wiping your disks clean.

The security issues are complex (and most are beyond the scope of this course). Some of the things to worry about are:

- Remote web users browsing beyond the confines of the web document tree. Especially being able to peruse such files as the system password file, user's home directories, system configuration files *etc.*
- Unauthorized local users knowingly or unwittingly modifying web documents, configuration files and/or directory access files.
- Remote crackers subverting the web site by exploiting bugs in the server or (much more likely) CGI scripts.
- Internet sniffers capturing network packets that contain sensitive information, such as username/passwords or credit card information.

Two types of tools for countering these threats are at the administrators disposal:

- security features built into the web protocols, and
- general network security measures that can be used to protect the server's host machine.

We will concern ourselves only with the former - how to use the web protocols to make the site as secure as these protocols allow. Securing the host machine is a topic in itself that will not be covered in this course.

12.2 Insecure Server Features

There are a number of basic precautions to take within the server software. The simplest of these is to turn off server features that won't be required. It is an axiom of programming that the simpler it is the less chance for mistakes.

12.2.1 Automatic Directory Listing

During the creation and maintenance of a site, all sorts of detritus can accumulate in the document tree: old documents, test CGI scripts, editor auto-save files, new documents not ready for inclusion into the document tree, and things that just seemed like a good idea at the time.

If automatic directory listings are left on, it is possible for a visitor to browse through your document tree perusing resources not meant for public consumption, and learning more about your system than you might like.

A standard and not very secure form of simple web security is to create documents in the document tree that no other document references — "orphan" documents. This means that the only way someone can view the document is to know its URL explicitly. This of course can only work if automatic directory listing is turned off.

There are two ways to turn off automatic directory listing

(a) Ensure that every directory has a *welcome page* so that it is displayed instead of the synthesised directory listing, or better yet,

(b) turn off the feature completely in the server configuration file.

The second option is by far the safest method, rather than remembering to ensure that every directory has a welcome page.

If automatic directory listing is required it can be turned on, on a directory by directory basis using the access control file for each directory.

Exercise 12.1: If your server has automatic directory listing turned on experiment with turning it off for specific directories, using the directory's access control file.

What happens if you request a directory where automatic directory listing is turned **off** and there is **no** welcome page?

12.2.2 Symbolic Links

Using symbolic links (also called "shortcuts") to extend the document tree to other parts of the file system is a potential security risk. When a web site is under the control of a number of people, it's easy for someone to inadvertently create a link to a sensitive place on the system (say */etc*, which contains such juicy items as the system's configuration files!).

If *symbolic link following* is turned off, the document tree can still be extended to other parts of the file system, but it must be done explicitly in the server's configuration file using an `Alias` directive.

Exercise 12.2: To create a symbolic link in Unix you use the command

```
ln -s [target-name] [link-name]
```

For example

```
ln -s /usr/man man
```

creates a symbolic link `man/` in your directory tree that points to `/usr/man/`. Changing directory to `man` places you in the directory `/usr/man/`.

Experiment with this command making symbolic links to your own files/directories and to files/directories you do not own (such as `/usr/man/`).

By modifying the server configuration file or the directory access files experiment with the `Options` directive and the parameters

`None`, `FollowSymLinks`, `SymLinksIfOwnerMatch`.

Don't forget to turn on automatic directory indexing if the target directories do not contain welcome pages.

How do the directives `FollowSymLinks` and `SymLinksIfOwnerMatch` differ? What does "...if owner matches" mean?

Note Be careful, you could compromise the security of the machine you are on if your create symbolic links to sensitive files.

12.2.3 CGI Scripts

Executable scripts pose the greatest risk because buggy scripts (and there are a lot of them out there on the Internet) can be coerced into doing things that their authors did not anticipate. The choices are to turn off scripts entirely (not very practical) or to be very careful.

Make sure that any script you write or install does not allow a well meaning or malicious user damage your system or access sensitive parts of your system (such as a password file!).

Exercise 12.3: What does an "SQL-Injection Attack" mean? How is it combated in PHP?

When the web server runs a CGI script it is run as the user id that has been assigned to the web server. Make certain that the web server is assigned a user id with **no** privileges.

Note PHP is a particularly dangerous scripting language. Mainly because it provides programing interfaces into so many areas of the underlying operating system. When installing the PHP module great care must be taken to ensure that only required features are turned on.

Exercise 12.4: Read the Security section of the PHP manual and familiarise yourself with some of the security issues of PHP.

12.2.4 User Directories

User directories are a major security issue. On multi-user systems where anyone can place web pages in their private area the potential for disaster is huge. For example

- Symbolic links to sensitive parts of the files system
- Insecure executable scripts
- Server side include documents running `exec` or `include` or just including sensitive documents.
- Inadvertently placing private documents into their public web directory.

Apart from the latter point if user directories are to be supported then in those directories server side includes, following symbolic links, executing scripts should **all** be turned off. For example:

```
<Directory /home/*/public_html>
  AllowOverride FileInfo AuthConfig
  Options Indexes SymLinksIfOwnerMatch IncludesNoExec
  <Limit GET POST>
     Order allow,deny
     Allow from all
  </Limit>
  <Limit PUT DELETE>
```

```
    Order deny,allow
    Deny from all
  </Limit>
</Directory>
```

User directories can be disabled in Apache by the directive

```
UserDir disabled
```

12.2.5 Access Control Files

The default name for the directory access control file is (with Apache) *.htaccess*. It is surprising how many sites have not renamed the file to something different. As the server needs to read this file to decide on the access control for the directory, once a user has gained access to the directory then the user may be able to download the access control file.

The access control file contains sensitive information about the web site. A diligent hacker (and there are a lot around) could map the entire site and all the access control files and exploit weaknesses in the document tree.

A simple security measure - change the name of the access control file from the default!

Exercise 12.5: By default the Apache Web server is configured not to allow the downloading of the *.htaccess* file. How is this done? What is the configuration directives used?

12.2.6 Log Files

A vast amount of information is stored in a Web server's log files. In fact all the information that is passed by the web client to the server in the HTTP header lines can be stored in a log file. Server scripts that fail, write error messages into the server log files, these are useful for debugging server scripts but could be used to exploit weakness in server scripts.

Most of the emphasis in this section is on how to make the server secure from internal mistakes or external attacks. There is another aspect to security ensuring that people who visit your site are ensured confidentiality. After all, the log files will contain complete information on every page retrieved by a particular host. If the user has had to log onto the site then the user name could be stored along with the host name.

Log files are sensitive documents that should be kept in a safe place, with restricted access. The information logged into log files (which is configurable in Apache) should be the bare minimum required to monitor your site and correct problems.

12.2.7 File Permissions

File permissions are a powerful way of restricting access to sensitive parts of the files system. If the web server cannot read a file, neither

can the world!

Exercise 12.6: The Change Mode command `chmod` is used in Unix to change file permissions on files and directories. After reading the Man page for `chmod`, you should be able to change the mode on files so that the `user` (that is the owner of the file) cannot read the file

```
chmod u-r filename
```

To reverse the mode, so that the user can read the file the command is

```
chmod u+r filename
```

If you can't read the file then neither can your server, as your server is running under your UID. Check that this is the case. What is the error status and message returned by the server?

How do directories need to be changed to restrict access to the files they contain, without changing the access permissions on those files?

12.3 Server Security Features

Universal access was the original driving force behind the web. The increase in web diversity has meant that security has had to become an integral part of the HTTP.

The type of security offered by the HTTP and web servers ranges from restriction based on domain names through to document encryption, and server/client encrypted digital signatures.

Security features on a Web server can be roughly split into three categories: Authentication, Authorisation and Communication—which can be used together or separately. Authentication requires a visitor to identify themselves with a username and password. If they pass that hurdle they are then checked to see if they are authorised to access the requested resource. Authorisation can be based on the authenticating username, the client IP, or domain name.

12.4 Authorisation Features

Authorization grants access to a resource based on the user's username or the machine the user is sending the request from.

12.4.1 IP/Hostname Access Control

In this type of resource authorisation, the server examines the incoming connection and grants or denies access to the resource based on the client's host name or IP address.

This form of authorisation would appear safe at first but there are number of holes that you should be aware of

- Host name look-ups are easily fooled by a technique known as *DNS spoofing*. In effect, the connecting machine masquerades as a trusted machine.
- IP address restriction is safer, it is harder to *spoof* an IP address, but not impossible.
- Restriction by Domain Name or IP address says nothing about the person trying to connect through the trusted machine. If the trusted machine has been broken into then the user might not be trusted! The compromised machine can be used as a base to compromise other machines including yours!

When a browser attempts to access a URL that has been placed under access restrictions, such as restricting the document to trusted hosts, the server checks that the request came from one of the allowed hosts. If it didn't the server returns to the client the header containing the *403 Forbidden* status code and will not service the requested URL.

12.4.2 Configuring IP/Hostname Access Control

To protect a directory based *only* on the IP address or host name of the connecting host the following example should be used as a basis.

Example 12.7:

```
<Directory /usr/local/web/private>
Order Allow,Deny
Allow from .uq.edu.au
Allow from .anu.edu.au
Allow from 127.75.63
</Directory>
```

The example above shows basic IP/hostname authorisation as it would be found in the server global configuration file. That is, the directory it is to apply to has been specified using the `Directory` directive. Authorisation directives can also be used within directory access control files, where the `Directory` directive is not required.

Note

Remember that directory access files can only override their inherited access control directives if the `AllowOverRide` directive of the parent directory has been set to either `All`, or `Limit`. See module 11 on server configuration.

The main directives for restricting access to resources are:

`Order ...` The order in which to evaluate the the restriction directives. For example,

```
Order Deny,Allow
```

`Deny from ...` Deny access to some domains. For example,

```
Deny from .ozemail.com.au
Deny from .com.nz
```

`Allow from ...` Allow access from some domains. For example,

```
Allow from zeus.usq.edu.au
Allow from .com.au
```

Limit directive

The `<Limit...>` and `</Limit>` directives establish the access policy for each HTTP method within a directory. Within the `<Limit...>` directive is specified the list of methods that *this* access policy restricts. The methods are the HTTP methods, such as `GET`, `POST`, `PUT`, `DELETE` etc. Clients that try to use the listed method will be restricted according to the restrictions listed within the `Limit` section. Ordinarily only the `GET` method need be restricted. The `POST` method needs to be used in directories that contain CGI scripts.

Note

If the `Limit` directive is missing then the access directives apply to **all** methods. As the number of methods increases (there are many more than discussed in this study book) it is better to leave out the `Limit` directive and ensure all methods are restricted.

Deny and Allow directives

Each `Deny from` and `Allow from` directive list either the IP address or the hostname of a machine to either *deny* access or to *allow* access.

Each listed host can be a fully qualified domain name, such as `www.sci.usq.edu.au`, a partial domain name such as `.uq.edu.au`, a full IP address, such as `139.23.45.67`, a partial IP address, such as `139.23`[1]; or the keyword `all`, signifying *all* hosts trying to connect.

Hosts can be listed on one long line, separated with spaces, or in multiple short directives.

Order directives

The order in which the `Allow` and `Deny` directives are processed is important, because it is the first match which is used. The `Order` directive controls this.

Note

The order in which the `Allow` and `Deny` directives are processed is **not** the order they appear in the file. But the order specified by the `Order` directive.

The possible values of the `Order` directive are:

`Order Deny,Allow` The deny directives are evaluated before the allow directives.

All requests that do not match any Allow or Deny directives are permitted by default!

`Order Allow,Deny` The allow directives are evaluated before the deny directives.

Any requests that do not match any Allow or Deny directives are denied by default!

[1] Remember that the order for IP addressing is the reverse of a fully qualified domain name. IP addressing is left-to-right while fully qualified domain names are right-to-left.

When an `Order` directive is processed by the server the fall through or non-matching state for the evaluation of the `Deny/Allow` directives is set. If the `Deny` directives are to be processed first then the non-matching state is to *allow all* machines to connect. If the `Allow` directives are to be processed first then the non-matching state is to *deny all* machines from connecting.

The non-matching state is equivalent to an implicit "`Deny from all`" directive or an "`Allow from all`" directive.

Note

If you are uncomfortable relying on the implicit non-matching state an explicit `Deny from all` or `Allow from all` should be added to the access directives.

Exercise 12.8: If there was not an initial state set what would/could happen when processing the `Deny/Allow` directives?

Example 12.9: Here is an example of allowing everyone into your site except for a number of machines that have been giving you trouble

```
Order Allow,Deny
Allow from all
Deny from 139.34.128.67
Deny from ozemail.com.au
```

Exercise 12.10: In the example above, what would be the result if the `Order Allow,Deny` directive was changed to `Order Deny,Allow`?

Example 12.11: Here is an example of denying access to everyone except for a few trusted machines

```
Order Deny,Allow
Deny from all
Allow from .sci.usq.edu.au
Allow from 203.45.213.2 203.45.45.5
```

Exercise 12.12: In the example above, what would be the result if the `Order Deny,Allow` directive was changed to `Order Allow,Deny`?

Exercise 12.13: In the example above, what would be the result if the `Deny from all` directive was removed? How could you fix this problem without putting the `Deny from all` directive back in?

Exercise 12.14: An administrator has set up the following access control

```
Order Allow,Deny
Deny from all
Allow from .sci.usq.edu.au
```

What is the result?

There are two ways to fix this problem—what are they?

Exercise 12.15: An administrator has a directory containing HTML pages that are available to the world. This directory also has a CGI scripts that should only be accessed from a specific secure network.

How would the administrator set up the access control file for this directory?

Test your solution on your personal web site.

Exercise 12.16: Create a directory and place in it an access control file that denies access to the machine you are coming in on.

If you do not know your hostname use the Unix command `hostname` to find out.

Make sure it is closed to you by using your browser to request a document.

Connect to your server using `telnet` and request a document from that directory.

What is the server's HTTP response?

What happens to the HTML code in the body of the response?

Exercise 12.17: In Apache IP/Hostname access control is implemented by the module `mod_authz_host`. If you study the manual pages for this module you will notice that the `Allow from` and `Deny from` directives have the extra parameter, not discussed here, called `env`.

Using the examples in the Apache documentation, implement the directives "`User-Agent`" and "`Deny from env=...`" so that you are denied entry based on the browser you are using!

The module `mod_setenvif` contains directives that can be used to pass information to scripts and to other server modules so that responses can be modified based on information contained in the HTTP headers of the request.

12.5 Authentication Features

Server authentication requires the clients to authenticate themselves first using a username/password. If they fail to authenticate themselves the request is rejected with a `401 Unauthorized` response from the server. If the authentication is successful the user is then checked to see if they are authorised (see §12.4 below) to access the resource. If that is successful then the resource is sent.

12.5.1 User Authentication

Currently there are two schemes employed by clients and servers for *username/password* authentication, this is the `Basic` scheme and the `Digest` scheme. Both schemes, apart from requiring a username and a password also specify a *realm name*. The realm name is sent to the client when the server is requesting username authorisation ("401

Unauthorized" status response from the server). The client places the realm name into the dialog window where the user enters her username and password. On sites where a single user is required to enter a different username/password to be able to access different parts of the document tree the realm name is used to tell the user and client which username is expected.

Most browsers are smart enough to remember the username/password pairs that a user has entered during a session. The username/password pairs are associated with *realms.* This means that whenever you re-enter a *realm*, the browser will supply the correct username/password for that *realm.*[2]

When the `Basic` scheme is used the username/password are encoded using Base64 encoding, this is equivalent to sending the pair in clear text. When the `Digest` scheme is employed the password is not sent in clear text by the web client but is sent as part of an MD5 hashed string (see §12.6.3). The string is made up of the realm, username, password, method, URI and a one off random string sent by the server, the *nonce* (*number used once*).

12.5.2 Configuring User Authentication

Adding password protection to a directory requires some preparation. A list of authorised users needs to be created. Each user needs to be assigned a password. Then the directory access file (or the global configuration file) needs to be modified so that only selected users are authorised to access the documents in the directory.

Username/Password Files

The first step in setting up user authentication is creating a file of username/passwords. Apache offers many possible variants of this file. For example, username/passwords could be stored in a text file, a database file, the system file could be used, a remote server can be queried &c.

The original system offered by servers (and the Apache default) is a human readable file which is suitable for storing up to a hundred username/passwords. This file is constructed completely independently of the host operating system. Passwords are stored encrypted either using the Unix crypt algorithm or the MD5 algorithm, both are one way hash algorithms.

The Apache distribution comes with a utility programs called `htpasswd` or `htdigest`. These program will create and add users to a password file that Apache is capable of using to authenticate a user. For basic authentication the `htpasswd` program is used, for digest authentication the `htdigest` program is used.

The command line parameters for `htpasswd` are

`htpasswd [-c]` *password_file user*

[2] The only way to force most browsers to forget username/password pairs for realms is to shut them down!

password_file is the path to the password file to be modified. *user* is the name of the user to add to the password file. The *optional* argument `-c` is used *only* when you want to create a new password file.

The command line parameters for `htdigest` are

`htdigest [-c]` *digest_file realm user*

digest_file is the path to the digest file to be modified. *user* is the name of the user to add to the digest file. The *optional* argument `-c` is used *only* when you want to create a new digest file.

Example 12.18: The following is an example of creating a new password file named *password* and adding the user *Zaphod* to it.

```
>cd ~/CSC2406/secure
>htpasswd -c password Zaphod
Adding password for Zaphod
New password: ********
Re-type new password: ********
>
```

The example above created a new password file called `password`. There is nothing special about the name, it could have been anything and there was nothing special about the location, it could have been anywhere. Except you should never place a password file anywhere within the document tree otherwise a remote user could download it!

The password file created by the *htpasswd* program is a text file that looks something like the following

Example 12.19: An example of the contents of the password file created by *htpasswd*

```
Zaphod:Xvqw73TD/a1
Trillian:ghL/POnQ20jK
Ford:q109F3TmnP
Arthur:lPkj198Bvd7a
```

Each line contains a "username:password" pair. The password encrypted using the Unix crypt algorithm.

The username appears at the beginning of each line, followed by a colon and the user's encrypted password (no spaces surround the colon). The password has been encrypted using the Unix *crypt* routine. This file can be modified by a text editor. The editor can be used to fix spelling errors in users' names or deleting users entirely.

Example 12.20: An example of the contents of the digest file created by *htdigest*

```
Zaphod:Milliways:50682dbf5c2f76feaa59e3a57c145e35
Trillian:Milliways:bf7404ee0811050bffbad96917fcf5f9
Arthur:Milliways:405dfe0cbd38eac801ed2ad6f822ca86
Ford:Milliways:21dae74dd78e35a6183a439c3f53bda1
```

Each line contains a "username:realm:password" triplet. The password encrypted using the MD5 algorithm.

Exercise 12.21: Use the *htpasswd* program to create a number of username/passwords for your site.

Information on the Unix *crypt* routine can be found through the man pages. This routine is used to encrypt the user passwords on all Unix machines. Have a look at the man page for this routine.

Exercise 12.22: Use the *htdigest* program to create a number of username/passwords for your site.

Note

DO NOT put the password or digest file anywhere in the document tree. Any document in the document tree can *potentially* be downloaded by remote users.

An obvious place to store authentication files is somewhere within the *server root*, outside the document tree.

Group Files

Apache supports the idea of *groups*. If you have several distinct categories of user, each with their own authorization rights, administration can be simplified by creating a series of named groups. The information on groups is maintained in a *group file.*

There is no special tools for maintaining group files. They are simple text files containing a list of group names and the users assigned to each group. For example

```
# example of a group file
# Comments start with the "#" symbol

admin: george  anna
staff: anna george fred keith
students: w9712675 w9204826 w9837659 w8971230
CSC2406: w9712675 w9204826
MAT3100: w9837659
```

In the above example we have five groups, *admin, staff, students, CSC2406, MAT3100* with a varying number of users assigned to each group. Users can belong to more than one group simultaneously. The structure of the group file is similar to the password file - group name, colon, then a space separated list of users.

The group file can be placed anywhere it is convenient. Normally it is placed in the same directory as the password file. There can be multiple group files as there can be multiple password files. For large sites it makes sense to split up password and group files. If the site has more than a couple of hundred users then the simple text versions of the password and group files become too tedious to maintain and will slow the response of the server as it authenticates and then authorises access. In these cases it is better to create more efficient binary databases or relational databases of users and groups.

Note

DO NOT put the group file anywhere in the document tree. Any document in the document tree can *potentially* be downloaded by remote users.

An obvious place to store authorisation files is somewhere within the *server root*, outside the document tree.

Exercise 12.23: Create a group file for the users used in Exercise 12.21.

Authentication Directives

After the password and group files have been created, then the authentication directives can be placed either in the global configuration file, or in the directory access file. Combined with the authorization directives discussed above they provide a powerful system for restricting access to resources on a Web server.

Note

Remember that directory access files can only override their inherited authentication and authorisation directives if the `AllowOverRide` directive of the parent directory has been set to either `All`, `Limit`, or `AuthConfig`. See the section on server configuration.

The directives that control password authentication tell the server, where to find the password file, where to find the group file, the realm name of the directory, authentication scheme to use, and finally the authorization rules for the authenticated users.

Below is an example of a typical directory control section.

Example 12.24:

```
<Directory /usr/local/web/private >
# Add any options you want here
# Comments are started with the "#" symbol
AuthName  "Private Section"
AuthType Basic
AuthUserFile /usr/local/etc/httpd/secure/password
AuthGroupFile /usr/local/etc/httpd/secure/group

require group staff

</Directory>
```

This configuration example limits all access to only those users in the group `staff`.

Auth... directives

The Auth... directives set up the basics for username/password authentication. The `AuthName` directive specifies the "realm" name to use when requesting authorisation for this directory. The realm name is normally displayed in the browsers username/password dialog box.

`AuthType` specifies the type of scheme to use for username/password authentication. Either "Basic" or "Digest".

The *basic* scheme does not allow for the user's password to be encrypted by the browser before being sent to the server. The password is sent in clear text, which could be intercepted by someone. The *digest* scheme improves on this by sending the MD5 has of the password combined with a server generated string (the nonce).

The only way currently to ensure passwords are truly encrypted is to implement the application level Secure Socket Layer developed by Netscape Communications. This unfortunately is not a simple procedure and requires some knowledge of public/private key encryption, and certificate authorities (see §12.6).

The main authentication directives are:

`AuthName ...` Name the authentication realm. Must be in double quotes. For example:

```
AuthName "Members Only"
```

`AuthType ...` Authentication scheme. For example:

```
AuthType "Digest"
```

`AuthUserFile ...` Full path to the password or digest file. For example:

```
AuthUserFile /etc/httpd/passwd
```

`AuthDigestDomain ...` The domains this digest covers (only for Digest authentication). For example:

```
AuthDigestDomain /secure/
```

`AuthGroupFile ...` Full path to the group file

```
AuthGroupFile /etc/httpd/group
```

The `AuthUserFile` and `AuthGroupFile` directives give the full *physical* path of the password file and the group file. Although a `AuthUserFile` is required for user authentication, the `AuthGroupFile` is only needed if groups are going to be used for authentication.

The `AuthDigestDomain` directive is only used with digest authentication. It allows you to specify one or more URLs which are part of the same realm (See the manual for more information on digest authentication).

Require directive

This directive specifies which users and/or groups can gain access to the directory. Remember, in all cases the user must still supply the correct password to gain access! The require directive specifies which users are even allowed to try.

As with the "`order from`", "`deny from`" directives, the `require` directive can be used in a `Limit` section. If the `require` directive is not inside a `Limit` section then all HTTP methods will require authentication. Which in most cases is the behaviour you would want.

The form of the `require` directive is:

`require user name1 name2 ...` Only the named users can access the contents of the directory

`require group group1 group2 ...` Only users belonging to the named groups can access the contents of the directory

`require valid-user` Any user defined in the password file can gain access to the directory (as long as they know their password!)

Only one `require` directive should be used. Though any number can be used it is pointless as they all must match to authenticate the user.

Access control using IP/Hostname directives can be combined with username/password authentication. The default action when the two methods are combined is that the user must satisfy *both*, that is address and user authorisation. The `Satisfy` directive can be used to modify the default behaviour so that users are granted access if they are connecting from trusted machines or enter a valid username/password.

Exercise 12.25: Add an access control file to one of your directories in the document root. Allow access to users from your password file created above.

Modify the access file and play with various combinations of users and groups.

Exercise 12.26: Create multiple password files with different users in each. Assign a different password file to a parent and child directory. Use access control files for this.

Add an `AllowOverRide` directive to the parent directory. Play with various combinations of `All, None, AuthConfig` and `Limited`. What effects do these various combinations have?

12.6 Communication Security

With the advent of electronic commerce it was clear that if it was to succeed then it must be reliable, must ensure data integrity, and protect transactions against third-party threats. If all three cannot be assured then consumers should be unwilling to provide credit card payment information over the Internet (though most consumers appear blithely unaware of the dangers).

The three factors vital for the continuation of electronic commerce are:

Privacy The ability to control who sees, or cannot see, information and under what terms.

Authenticity The ability to know the identities of communicating parties.

Integrity The assurance that stored or transmitted information is unaltered.

The server security we have been discussing so far does not address the problem of a third party intercepting, reading or even altering the communication stream between client and server. That is, how do

we ensure that the communications between client and server are not susceptible to unauthorised network monitoring or *packet sniffing*.

12.6.1 Encryption

One solution is to encrypt the communication between client and server.

Before we continue we need to define a few terms:

Cleartext Data that can be read and understood without any special tools.

Encryption The method of disguising plaintext in such a way as to hide its content.

Ciphertext Cleartext data that has been *encrypted* to render it incomprehensible.

Decryption The process of turning *Ciphertext* back into *Cleartext*.

Cryptography Is the science, using mathematics to encrypt and decrypt data. Cryptography enables users to store sensitive information or transmit it across insecure networks (like the Internet) so that it cannot be read by anyone except the intended recipient.

Cryptanalysis While cryptography is the science of securing data, *cryptanalysis* is the science of analysing and breaking secure communication. Cryptanalysts are also called *attackers*.

12.6.2 Cryptographic Algorithms

A *cryptographic algorithm* or *cipher*, is a mathematical function used in the encryption and decryption process. A cryptographic algorithm works in combination with a *key* – a word, a number, or a phrase – to encrypt the plain text. The same plain text encrypts to different ciphertext with different keys. The security of encrypted data is entirely dependent on two things: the strength of the cipher and the secrecy of the key.

Symmetric Key Encryption

In *symmetric-key* or *secret-key* cryptography one key is used both for encryption and decryption. Most modern symmetric-key ciphers are known as *block ciphers*. This means that the cleartext message is encrypted in blocks. Historically the individual elements of a block where the characters of the message, with the introduction of computers into cryptography the individual elements of the block have become bits. The characters of the message are represented by their ASCII equivalents, and the message is chopped into fixed length blocks. Encryption is performed on each of the blocks using a key that is also a number.

There are a number of *symmetric-key* algorithms being used today, for example Data Encryption Standard (DES), triple-DES, Carlisle Adams and Staffard Travares (CAST) algorithm, International Data Encryption Algorithm (IDEA), Blowfish, ...

The size of the key, in *symmetric-key* cryptography is a crucial factor in determining the "strength" of a cipher. A cryptanalyst trying to decipher an encrypted message could attempt to check all possible keys, and the greater the number of possible keys, the longer it will take to find the correct one. The size of keys today is measured by the number of bits it takes to represent all possible keys, for instance the Blowfish, CAST and IDEA algorithms use a 128 bit key, the triple-DES a 168 bit key. The DES encryption, the official American encryption standard, with its 56 bit key is considered woefully inadequate for strong encryption by today's standards.

Symmetric-key cryptography, has major advantages:

- It is very fast.
- If the key is large enough, it is extremely difficult to crack.

Unfortunately it also has one major flaw. For a sender and recipient to communicate securely using a *symmetric-key* cipher, they must agree upon a key and keep it secret between themselves. If they are in different physical locations, they must trust a secure communications medium, such as a courier. Anyone who intercepts the key in transit can later read, modify, and forge all information encrypted or authenticated with that key. Ensuring the secure transmission of encryption keys is an expensive endeavour. In the 1970s financial institutions were expending large sums of money in secure courier services entrusted with carrying encryption keys around the world.

The persistent problem with *symmetric-key* encryption is *key distribution*: how do you get the key to the recipient without someone intercepting it.

Asymmetric Key Encryption

The problem of key distribution was solved by *asymmetric key cryptography* which is also known as *public key cryptography*. The idea was first put forward by J.H. Ellis, of the British Secret Service in the late 1960s, but was never made public (though the British did nothing with it). The idea was independently proposed in 1975 by Whitfield Diffie and Martin Hellman.

The idea behind public key encryption is to use a *pair of keys* for encryption: information encrypted with one key can only be decrypted by the other key—not by the encrypting key.

One key is made public (the *public* key) distributed to the world and the other key remains private and never given to anyone else (the *private*, or *secret* key). Anyone with a copy of the public key can then encrypt information that can only be decrypted by the holder of the corresponding private key. It is computationally unfeasible to deduce the private key from the public key (if they are long enough!).

With public-key encryption the need for sender and receiver to share secret keys via some secure channel is eliminated; all communications involve only public keys, and no private key is ever transmitted or shared.

Some examples of public-key crypto-systems are Elgamal (named for its inventor, Taher Elgamal, RSA (named for its inventors, Ron Rivest, Adi Shamir, and Leonard Adleman), and Diffie-Hellman.

Note

Anything encrypted by the private key can be decrypted by the public key. The encryption/decryption is symmetric—just the use of the keys is asymmetric.

Encrypting with the *private key* means a recipient can ensure only one person could have written it—as it can only be decrypted with that person's public key.

Example 12.27: The RSA algorithm is based on the fact that it is easy to multiply two large prime numbers together, but hard to factor them out of the product if they are unknown.

Key Generation
To create a public-private key pair using the RSA algorithm is relatively simple:

i. Generate two large prime numbers p and q.
For this example we will use two small prime numbers, to be secure two really large prime numbers are required.

We will use:

$$p = 197$$
$$q = 113$$

ii. Calculate

$$\begin{aligned} m &= (p-1)(q-1) \\ &= 196 \times 112 \\ &= 21952 \end{aligned}$$

iii. Calculate

$$\begin{aligned} n &= pq \\ &= 197 \times 113 \\ &= 22261 \end{aligned}$$

iv. Choose a small number d, that is relative prime to m.
When two numbers are relative prime, it means that the largest number that can divide both (their greatest common divisor) is 1.

Here we choose $d = 5$.

A simple method for finding d is to start at 2, incrementing by one each time until an appropriate d is found.

(*Euclid's Algorithm* can be used to find the greatest common divisor of two numbers quickly and easily, but the details are omitted here)

v. Find e, such that $de = 1 (\text{mod } m)$.
This means find an e such that when de is divided by m the remainder is 1. One way of finding e is to rewrite the the equation above as $de = 1 + nm$ where n is any integer. Rewriting this equation as $e = (1 + nm)/d$, we can work through integer values of n until we find an integer solution for e.

Doing that we find $e = 8781$.

(For large numbers a variation on *Euclid's Algorithm* can be used to quickly and easily find the e.)

vi. The keys are:

Public: (n,d)=(22261,5)
Private: (n,e)=(22261,8781)

At this stage the prime numbers q and p should be discarded.

Encryption
The message to be encrypted has to be a number less than the smaller of p and q. However, at this point p and q are unknown, and must remain unknown, so a number well below the minimum can be published, without compromising security.

To encrypt a number M, the encryption formula is

$$E(M) = M^d (\text{mod } n)$$

We will encrypt the character Q, which has the ASCII value 81.

$$\begin{aligned} E(M) &= 81^5 (\text{mod } 22261) \\ &= 3486784401 (\text{mod } 22261) \\ &= 21710 \end{aligned}$$

So the encrypted value of Q is 21710.

Decryption
To decrypt a number $E(M)$, the decryption formula is

$$M = E(M)^e (\text{mod } n)$$

So decrypting the value 21710 we need to calculate

$$\begin{aligned} M &= 21710^{8781} (\text{mod } 22261) \\ &= 1621473118948704 \ldots 00000000 (\text{mod } 22261) \\ &= 81 \end{aligned}$$

Raising 21710^{8781} is problematic, the solution is a number with 38634 digits! (See the examples directory for the complete number.) Use can be made of the fact that *modulo n* of a number is equal to multiplying together the *modulo n* of the factors of the number. For example:

$$\begin{aligned} & 12345 \times 123456 \times 1234567 (\text{mod } 11) \\ = \; & (12345 (\text{mod } 11) \times 123456 (\text{mod } 11) \times 1234567 (\text{mod } 11)) (\text{mod } 11) \end{aligned}$$

Public-key encryption seems to solve the main disadvantage of symmetric-key encryption, unfortunately public-key encryption is extremely slow! So most cryptographic systems combine the best features of both symmetric-key and public-key cryptography. The hybrid system becomes a multistage process to encrypt a message:

(a) Compress the cleartext document. Data compression saves space but more importantly, strengthens cryptographic security. Many cryptanalysis techniques exploit patterns found in the cleartext to break the cipher. Compression reduces these patterns, thereby greatly enhancing resistance to cryptanalysis. This step is not critical and can be bypassed.

(b) Generate a random *session key*, which is a one-time-only secret key. Using one of the fast and secure symmetric-key encryption algorithms with this one-time-only key, encrypt the cleartext message.

(c) Now the session key is encrypted using the recipient's public key.

(d) Send the encrypted data and the encrypted key to the recipient.

The advantage of this encryption strategy is that the large cleartext message is encrypted with the fast and secure symmetric-key encryption algorithm and the much smaller one-time-only random session key is encrypted using the much slower (about 1000 times slower) secure public-key algorithm.

12.6.3 Message Digest

A message digest is a compact "distillation" of your message. It is similar to a "file checksum". It can be thought of as a "fingerprint" of the message or file. The message digest "represents" your message, in such a way that if the message where altered in any way, a different message digest would be computed from it. Every unique string has a unique message digest.

The message digest is calculated using a one-way hash function. A one-way hash function takes a string of variable length and produces a representational string of fixed length, the *hash value*. Typically the hash value is 128 or 160 bits long. Hash functions are chosen so that it is computationally infeasible to find two distinct messages that will hash to the same value.

The two most popular message digest algorithms are the Secure Hash Algorithm (SHA) designed by the American National Institute of Standards and Technology (NIST), and the MD5 algorithm by RSA. The SHA produces an 160 bit hash value and the MD5 an 128 bit value.

Example 12.28: The Linux system has an application called `md5sum` (see the man page). This program will compute the MD5 hash value for any file.

For example, the MD5 hash value of the sentence:

```
This program will compute the MD5 hash value for any file.
```

is in hexadecimal:

```
b4fe56571d9e2270f459aab6395f9155
```

Removing the fullstop at the end of the sentence, and the hash value becomes

```
4245600a96891fbb0fb018d4cea8d0cd
```

One character change has produced a completely different hash value.

When hash values are used to detect whether a message has been altered, they are called *Modification Detection Codes* (MDCs).

When hash values have been calculated with a secret key added to the message to be hashed they are known as *Message Authentication Codes* (MACs)

Exercise 12.29: Read the man page for the `md5sum` application. Experiment with the application creating MD5 hash values of supplied strings.

Most Linux software available for download from the Internet also has an MD5 checksum that can be downloaded as well. Why?

12.6.4 Digital Signatures

A major benefit of public key cryptography is that it provides a method for employing *digital signatures*. Digital signatures enable the recipient of a message to verify the authenticity of the messages origin, and also verify if the information is intact. That is, public key digital signatures provide *authentication* and data *integrity*. A digital signature also provides *non-repudiation*, that is the sender cannot deny the message.

Digital signature's make use of a feature of public-key cryptography: cleartext encrypted with the *private key* can be decrypted with the *public key*. The steps for adding a digital signature to a message are:

(a) Generate a hash value of the cleartext message.

(b) Encrypt the hash value with your private key.

(c) Append the encrypted hash value to the message.

(d) Send the message.

When the message and signature are received, the recipient

(a) generates a hash value of the received message.

(b) decrypts the your hash value sent with the message with your public key.

(c) compares the two hash values.

(d) If they match the message is genuine.

As long as a secure hash value is used, there is no way to take the signature from one message and attach it to another, or alter a signed message in any way. The slightest change in a signed document will produce a completely different hash value.

12.6.5 Digital Certificates

One issue with public key crypto-systems is that users must be constantly vigilant to ensure that the public key they are using does in fact belong to the person they are encrypting for. If someone where to post a phony key with the name and user ID of the user's intended recipient, data encrypted to— and intercepted by—the true owner of the phony key is now in the wrong hands. This type of an attack is called, a `man-in-the-middle` attack.

In a public key environment, it is vital that you are assured that the public key to which you are encrypting data is in fact the public key of the intended recipient. *Digital Certificates*, or *certs*, simplify the task of establishing whether a public key belongs to the purported owner.

A digital certificate is information included with a person's public key that helps others verify that a key is genuine. Digital certificates are used to thwart attempts to substitute one person's key for another.

A digital certificate consists of three things:

(a) A public key.

(b) Certificate information. That is, identity information about the owner of the public key, such as name, user ID, organisation etc.

(c) One or more digital signatures.

The purpose of the digital signatures on a certificate is to state that the certificate information has been attested to by some other person or entity. The digital signature does not attest to the authenticity of the certificate as a whole; it vouches only that the signed identity information goes along with, or is bound to, the public key.

Implicit in certificates is the idea of a *trusted third party*, someone who will sign the certificate. The trusted third party is known as a *Certificate Authority* or *CA*, which is a person, group, department or company, that is trusted to verify the integrity of the contents of the certificate before signing the certificate with their private key.

Some internationally recognised certificate authorities are RSA, Verisign and Thawte

12.6.6 The Transport Layer Security Protocol

All of the features of modern cryptography, symmetric keys, public keys, hash values, and digital certificates, are employed to ensure secure communications between the web client and the web server. The Transport Layer Security (TLS) (which is based on its predecessor the Secure Socket Layer (SSL)) is the most widely used *secure* protocol.

TLS/SSL protects data in two steps. In the first step, the client and server perform a handshake. During the handshake process, they establish an agreed upon cryptographic methodology and exchange secret keys. Next, TLS/SSL takes application data and encrypts it. At the receiving end, this process is executed in reverse.

During the handshake process, the server sends the client its digital certificate. The client checks the signature of the CA on the server's certificate, if it does not belong to one of the predefined list of CAs the user is asked to accept or reject the server certificate.

If the certificate has been signed by a recognised CA or the user has accepted the certificate, the client generates a secret key and encrypts the key using the server's public key contained within the certificate. It then sends the encrypted key back to the server. Both client and server now have the same secret key that they use to generate the same secret symmetric-key and the same secret MAC key.

The data the server sends to the client is now compressed, encrypted using the agreed symmetric-key algorithm and secret key, and the server signs the data using a MAC. The client can reverse the process, and verify the data received. Data sent by the client to the server is also compressed, encrypted and signed.

The digital certificate ensures the identity of the server. The encryption of the data passed between the client and the server ensures privacy. Finally, the insertion of digital signatures into the data stream ensures data integrity.

Exercise 12.30: Under Linux the TLS/SSL is supplied by the OpenSSL package. OpenSSL is a library of routines that supply hash algorithms, symmetric key encryption algorithms, asymmetric key encryption algorithms and the routines for any application to communicate via TLS/SSL.

The supplied application `crypt` (see the course web site) can be used to encrypt and decrypt using the symmetric encryption scheme Blowfish.

Study the code especially the library file `encryption.c`, and experiment with the `crypt` application.

Why is the *pass phrase* you supply converted into an encryption key using MD5?

Why is the output from the Blowfish encryption encoded using Base64?

12.7 Questions

Short Answer Questions

Q. 12.31: What makes web sites vulnerable to malicious attack?

Q. 12.32: What is the major flaw in symmetric key encryption?

Q. 12.33: What are the security considerations for automatic directory listings?

Q. 12.34: What is access control?

Q. 12.35: What is a certificate authority?

Q. 12.36: Why is the order/deny order important?

Q. 12.37: Why do the order directives require an initial state?

Q. 12.38: Does someone in a group file need to be in the username file?

Q. 12.39: How is access granted to a directory based on groups?

Q. 12.40: What is the difference between symmetric key encryption and public key encryption?

Q. 12.41: What is the major flaw in public key encryption?

Q. 12.42: What is the difference between authentication and authorisation?

Q. 12.43: What is a hash value?

Q. 12.44: What feature(s) is it that the directive *Limit* is limiting?

Q. 12.45: Explain the concept of *groups* as it relates to access configuration.

12.8 Further Reading and References

(a) Read the section in the Apache Manual *Authentication, Authorization, and Access Control*

(b) The Apache documentation discusses alternative methods for user authentication. See *mod_auth* modules,