The World Wide Web

15.1 Introduction

The World Wide Web – normally abbreviated to "the Web" or sometimes "the Net" – is a vast collection of electronic documents each composed of a linked set of pages that are written in HTML. The documents are stored in files on many thousands of computers that are distributed around the global Internet. The concepts behind the Web were conceived in 1989 by Tim Berners-Lee when he was working at the European Particle Physics Laboratory, CERN. An agreement was signed in 1994 between CERN and the Massachusetts Institute of Technology, MIT to set up a consortium whose aim was to further develop the Web and to standardize the protocols associated with it. The National Center for Supercomputing Applications (NCSA) also made a major contribution to the current widespread use of the Web with the development of MOSAIC which was the first interactive Web browser based on a graphical user interface. Since that time, many related developments have taken place and, in terms of volume, the Web is now the largest source of data transferred over the Internet.

We presented an overview of the operation of the Web and the essential protocols and standards associated with it in Section 5.4. In this chapter we expand upon these descriptions as we discuss the following:

- **URLs and HTTP**: a URL comprises the name of the file and the location of the server on the Internet where the file is stored while HTTP is the protocol used by a browser program to communicate with a server program over the Internet;
- **HTML**: this is used to define how the contents of each Web page are displayed on the screen of the user's machine – a PC, workstation, or set-top box – and to set up the hyperlinks with other pages;
- **forms and CGI script**: these are used in e-commerce applications. Fill-in forms are integrated into a Web page and displayed on the screen of the browser machine to get input from the user and a CGI script is then used at the server to process this information;
- **helper applications and plug-ins**: these are used to process and output multimedia information such as audio and/or video that is incorporated into an HTML page;
- **Java applets**: these are separate programs that are called from an HTML page and downloaded from a Web server. They are then run on the browser machine. Typically, they are used for code that may change or to introduce interactivity to a Web page such as for games playing;
- **JavaScript**: this is also used to add interactivity to a Web page but in this case the code is not a separate program but is included in the page's HTML code;
- **security** in e-commerce applications;
- **the operation of the Web** including the role of search engines and portals.

In relation to HTML and Java/JavaScript, since there are now many books on each of these topics, the aim here is to give sufficient detail for you to build up a working knowledge of them. Further details can then be found in the bibliography for this chapter.

15.2 URLs and HTTP

The Web is made up of a vast collection of documents/pages which are stored in files located on many thousands of (server) computers distributed around the global Internet. As we saw in Section 2.3.3, using HTML it is possible to create on a server an electronic document in the form of a number of pages with defined linkages between them. A user then gains access to a specific page using a client program called a browser which runs on a multimedia PC/workstation/set-top box that has access to the Internet.

Associated with each access request is the uniform resource locator (URL) of the requested file/page. This comprises the domain name of the server computer on which the file/page is stored and the file name. To obtain a page the browser communicates with a peer application process in the named Web server computer using the HyperText transfer protocol (HTTP). The contents of the named file are then transferred to the browser and displayed on the screen according to the HTML markup descriptions the page contains. A schematic diagram showing this overall mode of operation is given in Figure 15.1. In this section we describe first the structure of URLs and then the operation of HTTP. We defer how a URL is embedded into an HTML page until section 15.3 when we describe HTML in more detail.

15.2.1 URLs

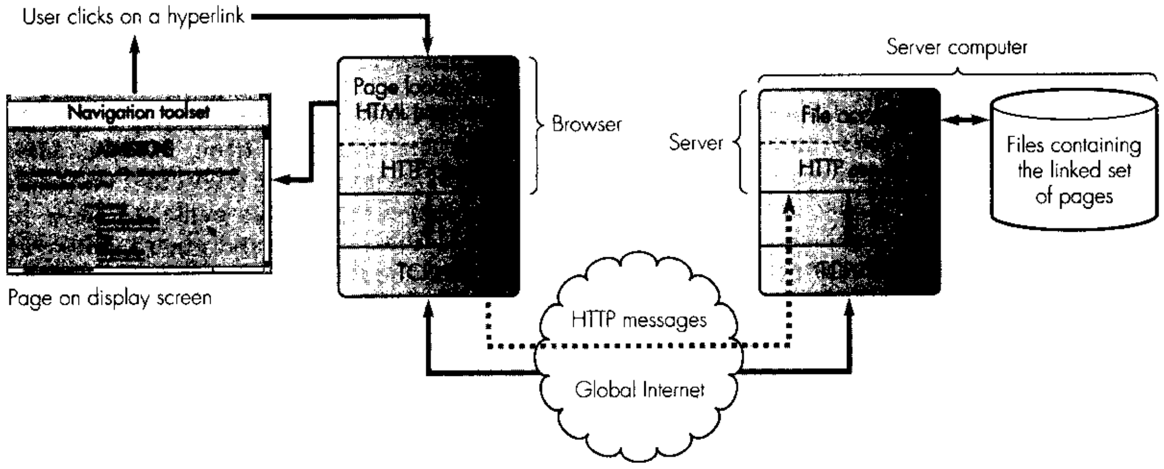
The standard format of the URL of an HTML page consists of:

- the application protocol to be used to obtain the page,
- the domain name of the server computer,
- the pathname of the file,
- the file name.

Thus an example URL referring to an HTML page on the Web is:

http://www.mpeg.org/mpeg-4/index.html

where *http* is the protocol used to obtain the Web page, *www.mpeg.org* is the domain name of the server, */mpeg-4* is the path name, and *index.html* is the file name.



Note: Hyperlinks contain a URL which includes the domain name of the server computer and the name of the file containing the HTML code of the selected page

Figure 15.1 Basic principles and terminology associated with the World Wide Web.

The main services offered by a browser were identified earlier in the book in Figure 2.9. Normally, a browsing session starts by a user entering the URL of the home page associated with a particular document in the *location* field provided by the browser. If the URL of the page is not known then it can be obtained either from the user's own local Web/Net directory – which is built up from previously given or previously used URLs – or by using the search facility supported by the browser. We shall expand upon this feature later in Section 15.7. Once a URL has been entered, the browser proceeds to access the (home) page from the server named in the URL using the specified protocol and the given file name.

Note that when we access most home pages it is not necessary to specify the full URL since the server will look for the file named *index.html* if one is not specified. For example, the home page associated with the above URL could be specified as:

http://www.mpeg.org

Note also that a final forward slash is used to indicate the URL relates to a directory rather than a file name. For example:

http://www.mpeg.org/mpeg-4/

In addition to obtaining a page/file using HTTP, most browsers allow a user to obtain a file using a range of other application protocols. In general, these are standard Internet application protocols that predate the Web. For example, as we saw in Section 14.4, FTP is the standard Internet application protocol used to transfer a file. As a result, many servers still use FTP for all file transfers. Hence to obtain the contents of a file from such a server it is possible to specify *ftp:* as the protocol in a URL instead of *http:*. An example of a typical URL is then of the form:

ftp://yourcompany.com/pub/

Typically, this file will contain the list of publications/files – note that *pub/* indicates a directory – that are available from the file server *yourcompany.com.* Normally, as we saw in Section 14.4.6, a user logs on to most public domain FTP servers using *anonymous* for the user name and his or her email address for the password. Hence these are entered when requested by the browser. The browser then obtains the file using the FTP protocol and displays the contents on the browser screen.

A protocol name of *file:* is used to indicate the file is located on the same computer as the browser; that is, your own computer. This is a useful facility when developing a Web page since it allows you to view the contents of the page before you make it available on the Web. An example URL is:

file://hypertext/html/mypage.htm

Note that since some older versions of DOS allow only three characters in a file name extension, the final *l* of *html* is sometimes missing.

The *news:* protocol relates to an Internet application protocol defined in **RFC 977** called the **network news transfer protocol (NNTP)**. It is used to transfer the text-only messages associated with **UseNet** which is also called **NetNews**. This consists of a world-wide collection of **newsgroups** each of which is a discussion forum on a specific topic. Two examples are COMP – which has topics relating to computers, computer science, and the computer industry – and SCI which has topics relating to the physical sciences and engineering. A person interested in a particular topic can subscribe to be a member of the related newsgroup. A subscriber can then post (send) an article to all the other members of the same newsgroup and receive the articles that are written by all the other members of the same group.

To do this, a user agent similar to that used with SMTP is used. This is called a **news reader** and the protocol that is used to transfer the messages associated with UseNet is NNTP. Some examples of the request/command messages associated with NNTP are:

- LIST: this is used to obtain a list of all the current newsgroups and their articles;
- GROUP *grp*: this is used to obtain a list of the articles associated with the newsgroup *grp*;
- ARTICLE *id*: this is used to obtain article *id*;
- POST*id*: this is used to send article *id* to the members of a specified newsgroup.

The *news:* protocol enables a member to both read a news article and to post an article from within a Web page. An example of a URL relating to a newsgroup involved in the preparation of HTML documents is:

news:comp.infosystems.www.authoring.html

Note that in this case the two forward slashes following the colon are not required. When this URL is entered, typically, the news reader part of the browser responds by first obtaining the list of articles on this topic using the GROUP command and the NNTP protocol. It then displays the articles on the screen in the form of a scrollable list. The user can then click on a specific article in the list and the browser/news reader will obtain the article contents using the ARTICLE command and display this on the screen. Normally, each article has the email address of the author, his or her affiliation, and the date the article was posted. A similar procedure is followed to post an article using the POST command. Normally, the user is prompted by the news reader for, say, the name of the (local) file containing the article. The file contents are then sent using NNTP.

The *gopher:* protocol relates to an Internet application protocol called **Gopher**. The Gopher system is similar in principle to the Web inasmuch as it is a global delivery and retrieval system of documents. In Gopher, however, all items of information are text only. When a user logs on to a Gopher server a hierarchical menu of files and directories is presented each of which may have links to the menus on other servers. A user can then access the contents of a file or directory by clicking on it. The *gopher:* protocol enables the user of the browser to do this within a Web page.

The *mailto:* protocol is provided to enable a user to send an email from within a Web page. Typically, this facility is initiated by the user entering a URL with *mailto:* in the protocol part. This is followed by the email address of the intended recipient. Normally, the (email) user agent part of the browser responds by displaying a **form** containing the other (email) header fields at the top (to be filled in) and space for the actual message. A facility is then provided to initiate the sending of the mail which, typically, is sent either SMTP or, if the email server supports it, HTTP. An example URL is:

mailto:yourname@youruniversity.edu

Note that the two forward slashes following the colon are not required with this protocol.

Universal resource identifiers (URIs)

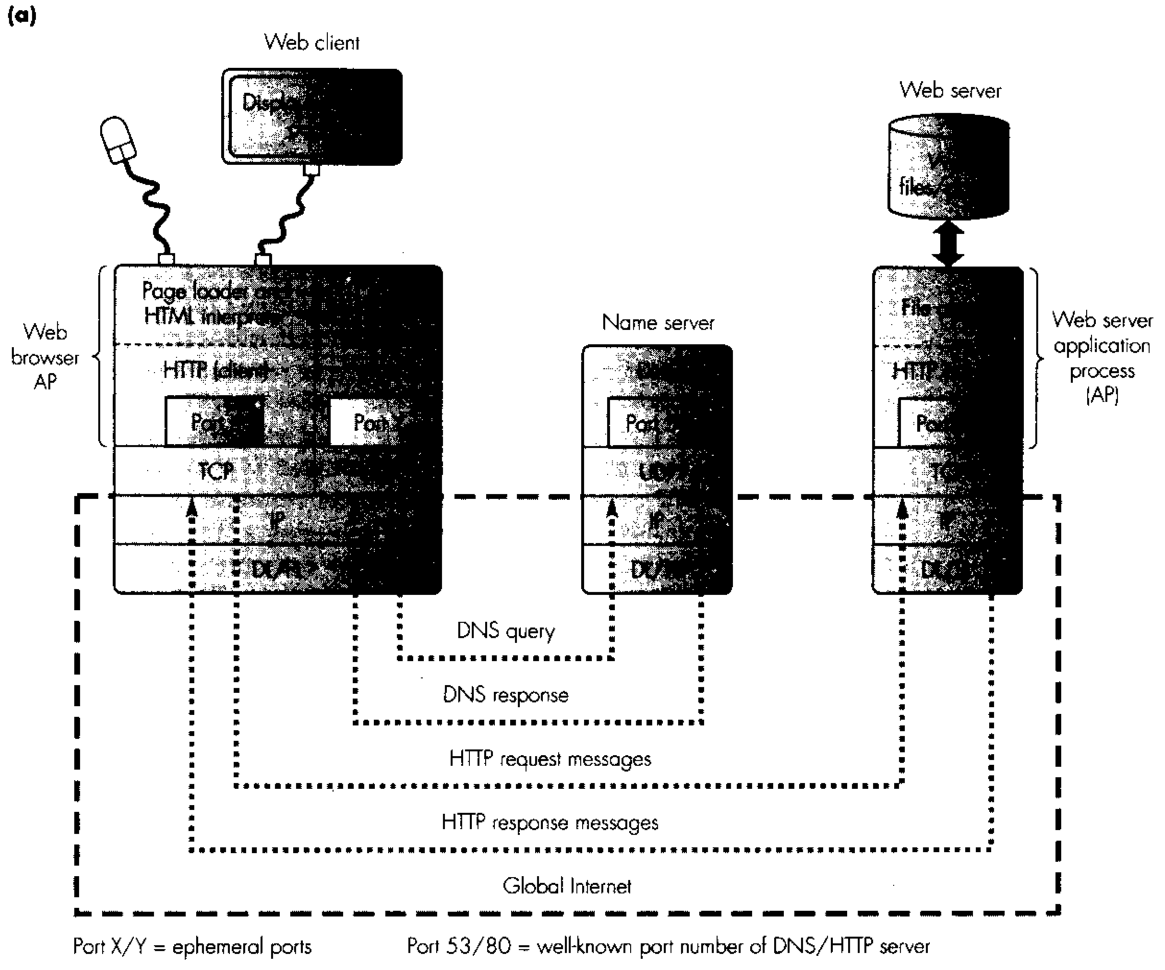
A limitation of a URL is that it specifies a single host name. In many instances, however, a Web site may have so many access requests/hits for a particular document/page that copies of the document must be placed on multiple hosts. Also, to reduce the level of Internet traffic, each of these hosts may be geographically distributed around the Internet. To enable this to be implemented, an alternative page identifier called a URI can be specified with some browsers. Essentially, this is a generic URL since, typically, it contains only the file name. The remaining parts of the URL are determined by the context in which the URI is given and these are filled in by the browser itself. We shall given an example of this in the next section when we discuss cache servers.

15.2.2 HTTP

HTTP is the standard application protocol – also known as a method – that is used to obtain a Web page and also other items of information relating to a page such as an image or a segment of audio or video. The earlier version of HTTP is defined in **RFC 1945** and a later version (1.1) in **RFC 2068**. It is a simple request-response protocol: the browser side sends a request message and the server side returns a response message. Figure 15.2(a) shows the protocol stack associated with Web browsing, and a selection of the *commands/methods* associated with request messages, together with their use, are shown in part (b) of the figure.

In general, these are self explanatory. Note, however, that the GET method is used also to, say, obtain an image or segment of audio – shown as a

(a)



(b)

HTTP	Use
GET <file name>	Read a Web page/block of data from the named file
HEAD <file name>	Read the header only of the specified Web page
PUT <file name>	Write a Web page or block of data to the named file
POST <file name>	Append a Web page/block of data to that in the named file
DELETE <file name>	Delete the named file

Figure 15.2 HTTP principles: (a) protocol stack; (b) a selection of the requests/methods supported.

block of data – from a named file. We shall expand upon this in later sections. Also, as we shall see in Section 15.3.6, the POST method is used to send the information entered by a user in e-commerce applications.

The well-known port number of HTTP is port 80. The Web server application process (AP) at each Web site continuously listens to this port for an incoming TCP connection request (SYN) from a Web browser. Note that in a Unix machine the AP is referred to as a **daemon** and the HTTP in these machines is sometimes called **HTTPD**. When a browser has an HTTP request message to send, it initiates the establishment of a new TCP connection to port 80 on the named server in the URL. It then initiates the sending of the request message over the established connection and, with earlier versions of HTTP, after the related response message has been received correctly by the browser, the server initiates the release of the connection. The TCP connection associated with this mode of operation is called **nonpersistent**.

As we can deduce from our earlier discussion of TCP in Section 12.3.2, the use of a new TCP connection for each request/response message transfer has a number of disadvantages. First, when accessing a Web page that contains multiple entities within it – an image for example – a time delay is incurred for each entity that is transferred while a new TCP connection is established. This is a function of the network round-trip time (RTT) which can be significant. Second, each new transfer starts with the slow start procedure and hence for a large entity, this can lead to additional delays.

To reduce the effect of these delays, when multiple entities are specified within a page – and hence to be transferred – many browsers set up a number of TCP connections so that each entity can be transferred concurrently. Typically, a browser may establish up to five or even 10 concurrent connections. However, although this can reduce the overall time delay associated with a Web access, the use of multiple connections leads to added overheads at both the client and server sides since both must maintain state information for each of the connections. This can be particularly significant for a popular/busy server which may get many hundreds of concurrent requests.

Hence with later versions of HTTP – version 1.1 onwards – unless informed differently, the server side leaves the initial TCP connection in place for the duration of the Web session. The TCP connection is then called **persistent** and, once in place, the browser may send multiple requests without waiting for a response to be received. Typically, the end of a session is determined by a timer expiring when no further transfers over the connection take place. Note, however, that the different versions of HTTP can interwork with each other.

Message formats

All HTTP request and response messages are NVT ASCII strings similar to those used with SMTP. With the earlier versions of HTTP – up to HTTP version 0.9 – what are called *simple request/response messages* are used. This means there is no type information associated with the request message which comprises only the method – GET, HEAD, and so on – followed by the related file name in the form of an ASCII string. The response message is in the form of a block of ASCII characters with no headers and no MIME extensions. An example of a simple HTTP request message is:

GET/mpeg-4/index.html

which is sent over the previously established TCP connection to the related server AP.

With the later versions of HTTP – version 1.0 onwards – MIME extensions are supported using what are called *full request/response messages*. To discriminate a full request from a simple request, a field containing the HTTP version number is added to the request line. This is followed by the text associated with a number of other RFC 822 headers, each of which is on a separate line. These were given earlier in Table 14.1 and include:

- general headers: these do not relate to the entity to be transferred and an example is the MIME version number;
- request headers: these are used to specify such things as the sender's name/email address and the media types and encodings that the browser supports;
- entity headers: these relate to the entity to be transferred and include the content type and, when sending an entity, the content length.

The end of the header is indicated by a blank line. Then, if the request contains a message body, this is followed by the entity being transferred such as a block of HTML text – a script – relating to a page.

The header fields associated with a full response message start with the HTTP version number followed by the response status code. Some examples are:

200	accepted
304	not modified: the requested page has not been modified
400	bad request
404	not found: requested page does not exist on this server.

This is followed by the name and location of the server and, if the response contains an entity in the message body, a *Content-Type:* and a *Content-Length:* field. Also, if the contents relate to a binary file, a *Content-Transfer-Encoding:Base64* header field.

An example showing a selection of the header fields associated with a full request/response message interchange is shown in Figure 15.3. The example relates to the transfer of the HTML page with the URL of:

http://www.mpeg.org/mpeg-4/index.html

Hence prior to sending the GET message, a TCP connection to the server *www.mpeg.org* will have been established.

(a) Example request message relating to a URL of

http://www.mpeg.org/mpeg-4/index.html

```
GET/mpeg-4/index.HTML HTTP/1.1
Connection:close
User-agent: Browser name/version number
Accept: text/html, image/gif, image/jpeg
```

(b) Example response message relating to this request:

```
HTTP/1.1 200 Accepted
Server: Aname
Location: www.mpeg.org
Subject: MPEG home page
Last-Modified: Day/month/year/time
Content-Type: text/html
Content-Length: 7684

Entity body comprising a string
of 7684 NVT ASCII characters
```

Figure 15.3 An example of a full request/response message relating to HTTP: (a) request message; (b) response message.

The meanings of the various fields in the request message shown in part (a) of the figure are as follows. The *Connection: close* header line indicates to the server that the browser does not need a persistent connection. The *User-agent:* line contains the name of the browser and its version number. Often the server contains a number of versions of a page and this enables the server to send the version that is best suited to the browser. The *Accept:* line indicates the entity types that the browser is able to accept which, as we can see, is determined by the compression software/hardware it supports.

The meanings of the various header fields in the response message shown in part (b) are mainly self explanatory. In the example, the body contains an entity comprising a string of NVT ASCII characters representing the HTML text of a Web page. Alternatively, if the *Content-Type:* was, say, *image/jpeg* then a *Content-Type-Encoding: Base64* header field would be present. For a more complete list of the MIME headers you should refer back to Table 14.1 and Figure 14.9 and their accompanying text.

Conditional GET

As we saw earlier in Figure 5.16(b), in many instances a browser does not communicate directly with the required server but rather through an intermediate system called a proxy server. In the figure it was assumed that the browsers at a site supported only the HTTP protocol and that the proxy server was used to access the contents of files using different protocols such as FTP and NNTP. As we can deduce from our discussion of URLs, this will avoid each of the browsers having the code of each of these protocols. In addition, however, a proxy server normally caches the Web pages and other entities that it obtains – on behalf of the browsers that it serves – on hard disk. Then, when a browser makes a request for a page/entity that the proxy server has cached, the proxy server can return this directly without going back to the server holding the original source. The latter is called the **origin server** and, when it performs this function, the proxy server is also known as a (Web) **cache server**.

Although caching reduces the response time for subsequent requests for a cached page/entity, there is a possibility that the cached entity may be out of date as a result of the original being modified/updated subsequent to the cache server receiving it. Hence because caching is widely used, an additional request message called a **conditional GET** is used by the cache server to ensure the response messages that are returned to the browsers contain up-to-date information.

A conditional GET request message is one which includes a header line of *If-Modified-Since:* and its use is illustrated in Figure 15.4. To avoid duplication, the header fields that have already been discussed are left out of the messages shown. The following should be noted when interpreting the message sequence.

- It is assumed that the browsers in all the client machines attached to the access network – site/campus LAN, ISP network, and so on – have been configured to send all request messages to the proxy/cache server.
- The sequence starts with a browser requesting an HTML page from the proxy server using a GET request message (1).

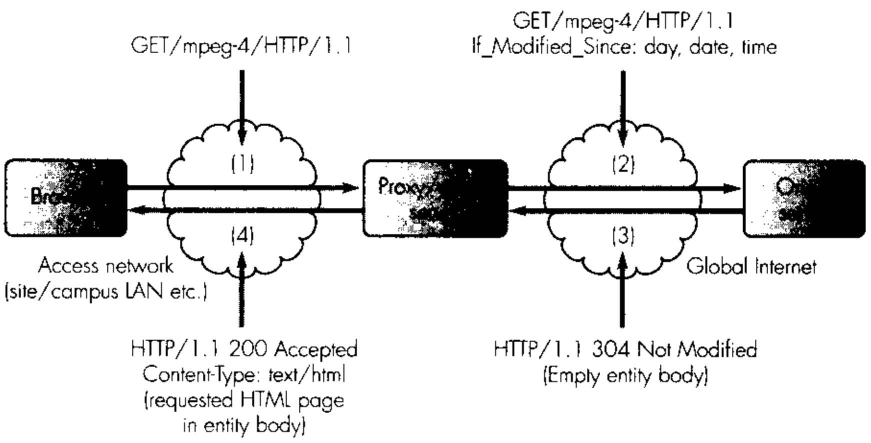


Figure 15.4 Proxy/cache server operation with conditional GET.

- The proxy server has a cached copy of the page and, associated with it, the day/date/time when the page was cached. This is obtained from the *Last-Modified:* header field in the response message returned by the origin server to an earlier request.
- Before returning the cached page to the browser, the proxy server sends a conditional GET request message to the origin server – defined in the page URL – with the date and time the current copy of the page it holds was last modified in the *If-Modified-Since:* header field (2).
- On receipt of this, the origin server checks to see if the requested page has been modified since the date in the *If-Modified-Since:* field.
- In the figure it is assumed that the contents have not been changed and hence the origin server returns a simple response message with a status code of *304 Not modified* in the header and an empty entity body (3).
- On receipt of this, the proxy server returns a copy of the cached page to the browser (4).

In the event that the requested page/entity had been changed, then a copy of the new page/entity would be returned by the origin server. A copy of the new page would then be cached by the proxy server – together with the date and time from the *Last-Modified:* field – before it is forwarded to the browser. Thus the savings obtained with a cache server come from the absence of an entity body in the response message from the origin server. Clearly for large pages/entities this can be considerable. In addition, further savings can be obtained by also having a higher-level cache server associated with, for example, each regional/national network. The proxy server associated with each access network is then configured to send all request messages to a specified higher-level cache server. In general, the higher the level this is in the Internet hierarchy the more requests it will receive and, as a result, the more cached pages/entities it holds.

15.3 HTML

The Hyper Text Markup Language, HTML, is the standard language used to write Web pages. As we saw in Section 2.3.3, HTML is the markup language that is used to describe how the contents of a document/page are to be displayed on the screen of the computer by the browser. Moreover, since the markup commands relate to a complete page, the browser automatically displays the page contents within the bounds allocated for the page; that is, irrespective of whether this is a small window on a low resolution screen or a large window on a high-resolution screen.

The markup/format commands are known as directives in HTML and the majority are specified using a pair of **tags**. In addition to the various types of directive associated with a string of text – which specify how the string is to be presented on the display – there are tags to enable a hyperlink to be specified as well as tags to specify an image or a segment of audio or video within a page and how these are to be displayed/output. There are also fill-in forms and other features. In this section we describe how each of these features is specified in HTML. It should be stressed, however, that HTML is continuously being revised and what follows should be considered only as an introduction to the subject.

15.3.1 Text format directives

The HTML text associated with a Web page is written in the **ISO 8859-1** Latin-1 character set which, for the English alphabet, is the same as the ASCII character set. However, for someone creating Web pages in a Latin alphabet, when using an ASCII keyboard, escape sequences must be used. For example, the Latin character *è* is represented by the ASCII string *&egrave* and *é* by *&eacute*.

The HTML text can be entered using either a word processor with facilities for creating and editing an HTML document/page or directly using the facilities provided by a Web browser. Typically, the complete string of characters consists of a number of substrings each of which may be displayed in a different format when the substring is output on the display by the HTML parser/interpreter part of the browser. In order for the interpreter to do this, each substring is sandwiched between a pair of tags that specify the format directive to be used. For example, if the substring

this is easy

is to be displayed in boldface, then this is written as

<B> *this is easy* </B>

As we can see, the opening tag comprises the format directive (B) between the pair of characters < and > and the closing tag is the same directive between </ and >. Note that the directive itself is case insensitive but normally upper-case is used to make the directives easier to identify. This format is used for a majority of the tags and a selection are listed in Table 15.1.

An example of an HTML script that includes some of these tags is shown in Figure 15.5(b). The example relates to the simple Web page we showed earlier in Figure 2.9 excluding at this stage the university crest. Typically, since this is the home page of the UoW, its URL would be:

hhtp:www.UoW.edu

The start of a page is indicated in the script by a <HTML> tag and the end of the page by a </HTML> tag. The <HTML> tag is followed by the page

Table 15.1 A selection of the HTML text format directive/tags.

Opening tag	*Closing tag*	*Use*
<HTML>	</HTML>	Specify the start and end of the complete document/page
<HEAD>	</HEAD>	Specify the start and end of the page header
<TITLE>	</TITLE>	Specify the title of the page. It is not displayed as part of the page
<BODY>	</BODY>	Specify the start and end of the contents of the displayed page
<Hn>	</Hn>	Specify the start and end of a level n heading
<B>	</B>	Display the related text in boldface
<I>	</I>	Display the related text in italics
<UL>	</UL>	Specify the start and end of an unordered/bulleted list
<OL>	</OL>	Specify the start and end of an ordered/numbered list
<LI>		Specifies the start of a listed item
<BR>		Specifies the start of a new line
<P>		Specifies the start of a new paragraph
<A HREF = "URL">	</A>	Specify an anchor/link to another page
<A HREF = "#NAME">	</A>	Specify an anchor/link to another point in the same page
<!--	-->	Start and end of comments

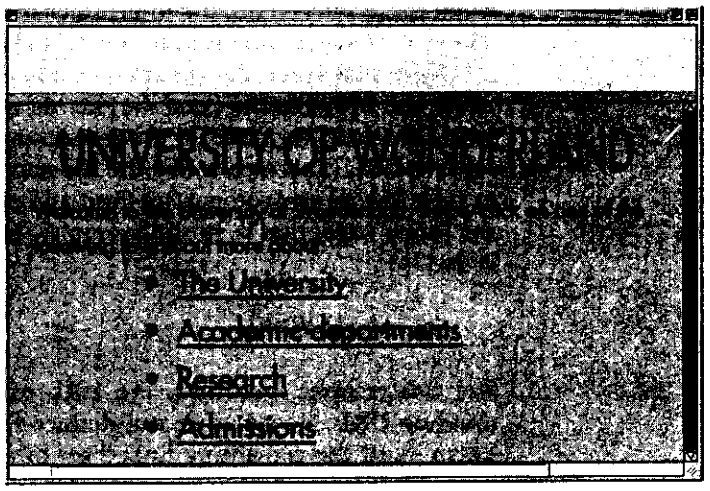
header which is entered between the <HEAD> and </HEAD> tags. Primarily, the header contains the title of the page which is written between the <TITLE> and </TITLE> tags. Note that the title is not output as part of the displayed page. In some instances, however, it is output by the browser in a field at the top or bottom of the display. The displayed page contents are then entered between the <BODY> and </BODY> tags.

In the example, there is only one heading and this is written between the <H1> and </H1> tags. If there were some subheadings, they would each be defined in the place they are to be displayed using the format:

<H2> Subheading name </H2>

Note, however, that it is the browser that decides the relative size of each heading/subheading on the display screen. Normally, they are in decreasing size with the first-level heading <H1> displayed in the largest font size and,

(a)



UNIVERSITY OF WONDERLAND

- The University
- Academic departments
- Research
- Admissions

(b)

```
<HTML>
<HEAD>
<TITLE> U of W Prospectus </TITLE>
</HEAD>
<BODY
<H1> UNIVERSITY OF WONDERLAND </H1>
<BR> Welcome to the University of Wonderland, simply click on one of the
following to find out more about:
<UL><LI><A HREF = "university"> The University </A>
     <LI><A HREF = "academic"> Academic departments </A>
     <LI><A HREF = "research"> Research </A>
     <LI><A HREF = "admissions"> Admissions </A>
</UL>
</BODY>
</HTML>
```

Figure 15.5 HTML text format directives: (a) example Web page; (b) the HTML script for the page.

typically, in boldface with one or more blank lines above and below it. Each level of subheading is then displayed in a decreasing size.

The <BR> is used to ensure that the welcome message starts on a new line. This is followed by an unordered (bulleted) list of items. These are listed between the <UL> and </UL> tags with each item starting with a single <LI> tag. The default for the start of the items in a first-level list is a solid bullet. In the example, each bulleted item is a hyperlink to a different page and hence starts with the <A HREF = "URL"> tag where A stands for **anchor**. This is followed by the textual name of the hyperlink – for example, The University – that is to appear on the display screen. Also, since this is a hyperlink name, the HTML interpreter displays it highlighted using, for example, an underline – as shown – or, if colored text is being used, a different color. Then, when this is clicked on, the interpreter uses the URL to fetch the related file. The end of a hyperlink is indicated by the </A> tag.

Note that since the prospectus consists of a linked set of pages, all of which are stored at the same site/server, once the full URL of the home page has been given, any links in the remaining linked set of pages can each use a **relative URL**. This means that the file names used in these URLs are assumed to have the same protocol/method and site/server domain name as that of the home page; that is, in the example, the browser assumes that each is preceded by:

http:www.UoW.edu/

For this reason, therefore, the URL of the home page is said to be an **absolute URL**. Hence as we can deduce from this example, by using relative URLs in all the remaining linked set of pages, it is then straightforward to relocate them by changing only the (absolute) URL of the home page.

In this example, the page containing the top-level index is relatively short and so fits into the display window. Also, it was assumed that further details relating to each of the listed headings were on a different page and accessed by an external link. However, if the index were larger and could not be displayed in a single window, then the user would have to scroll the page to find the remainder of the index. Alternatively, if the page/index is particularly long, it is sometimes preferable to have internal links within the page.

For example, if the second-level subheadings relating to each first-level heading shown in the example in Figure 2.9 were all on the same page, then the link associated with each first-level heading would be an internal link. These are specified using the format:

<A HREF "#University">The University</A>

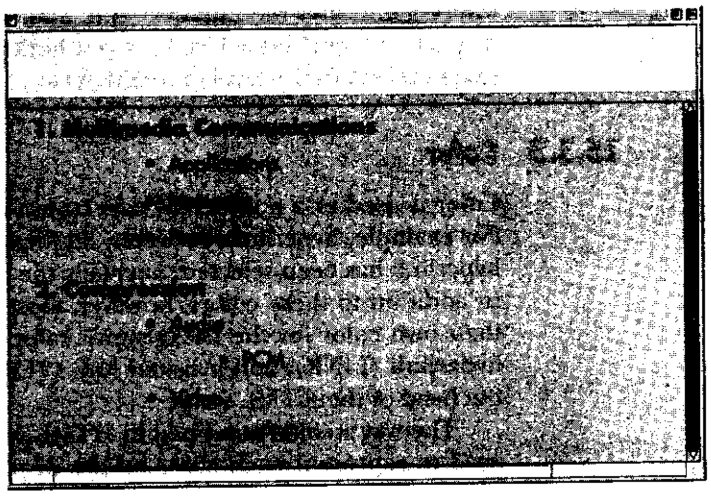
The words *The University* would then be displayed highlighted and, if the user clicks on this, the parser would jump to the point in the current page where the words *The University* next occur and start to display from this point in the page. The related subheadings could then be either internal links if further subheadings are defined or, if not, external links to the pages where the actual descriptions are located.

15.3.2 Lists

The example shown in Figure 15.5(b) contained a single unordered list. In addition, an ordered (numbered) list can be used as well as nested lists of different types. An example of an HTML script that illustrates these features is shown in Figure 15.6(b) and the resulting displayed page is shown in part (a).

As we can see, the listed items appear in the HTML script in the order they are to be displayed. The format directives associated with each item then enable the HTML interpreter in the browser to determine the position of

(a)



(b)

```
<HTML>
<HEAD><TITLE> Lists </TITLE></HEAD>
<BODY>
<OL><LI> Multimedia Communications
      <UL><LI> Applications
            <LI> Networks
            <LI> Protocols
      </UL>
      <LI> Compression
      <UL><LI> Audio
            <UL><LI>PCM</UL>
            <LI>Video
            <UL><LI>MPEG</UL>
</OL>
</BODY></HTML>
```

Figure 15.6 Lists in HTML: (a) example of displayed page; (b) HTML script for this page.

each item in relation to the others. Note that the default for an ordered list is a numeric value but it is also possible to define a number of alternatives. This is done by adding the required type to the opening tag:

```
<OL TYPE=X>
```

where X is the type of numbering. For example, X = i selects lowercase roman numerals (i, ii, and so on). It is also possible for a user to define the number he or she wants by using

```
<OL><LI VALUE=Y>
```

where Y indicates the number. Note that the default for the start of the items in a second-level unordered list is a hollow bullet and that it is the browser that determines the level of indentation.

15.3.3 Color

Color is used in a number of ways by browsers to enhance a displayed page. For example, hyperlinks are often highlighted in the color blue and, when a hyperlink has been selected, normally the color changes from blue to purple. In addition to these colors, however, most browsers allow the user to specify their own color for the background color (BGCOLOR), the text (TEXT), a hyperlink (LINK), and a visited link (VLINK) as part of the <BODY> tag at the head of the HTML script.

The way a color is defined in HTML is determined by the number of bits used to represent each color on the display of the machine the browser is running. As we saw in Table 2.1, this can range from 8 bits through to 24 bits. Normally, each 8 bits is represented by two hexacimal digits and hence each alternative color to be used is specified using either two (8 bits), four (16 bits), or six (24 bits) hexadecimal digits. For example, if 24 bits are used for each color, normally, the three sets of 8 bits represent the strength of each of the three primary colors (red, green, and blue) the color contains. The format used to represent the color is then:

#RRGGBB

where RR are the two hexadecimal digits that represent the strength of the color red, GG the color green, and BB the color blue. Hence the three primary colors are represented as:

#FF0000 = red, #00FF00 = green, #0000FF = blue

Some other examples are:

#FFFFFF = white, #FCE503 = yellow, #F1A60A = orange, #000000 = black

Hence if, for example, the background color is to be blue, the text is to be yellow, a hyperlink orange, and a visited hyperlink green, these would be specified as:

```
<BODY BGCOLOR = "#0000FF" TEXT = "#FCE503" LINK = "#F1A60A"
VLINK = "#00FF00">
```

Note, however, that most browsers use a color look-up table (CLUT) which displays only a defined set of 256 colors. Also, a number of these are reserved for the browser's own use. Hence although in theory a vast range of colors can be defined, in practice most browsers will simply approximate most of them to the nearest color-match in the usable colors in their CLUT. Typically, the usable colors that are available consist of any combination of the hexadecimal pairs:

```
00 33 66 99 CC FF
```

Hence when defining colors, if these hexadecimal pairs are used then the colors will be displayed in their unmodified form.

15.3.4 Images and lines

Images can be used both as an alternative to a white background for a page and for displaying a specific object within the page itself. Although the image can be in any format, most browsers support only GIF and JPEG images; that is, they only have the software to decompress an image held in either a *gif* or *jpeg* file. Horizontal lines in various forms can also be displayed on a page. We shall outline each feature separately.

Background images

In order to specify an alternative background image for a page, the file containing the image – for example *bgimage.gif* – is specified as part of the <BODY> tag using the format:

<BODY BACKGROUND = "*bgimage.gif*">

For example, most Web browsers have a file called *clouds.gif* which is often accessed and used as an alternative background. When the parser encounters this, the interpreter first obtains the contents of the file and, after this has been decompressed, displays this on the screen as the background. The page contents are then superimposed on it.

Images

An image that is displayed from within a page is specified at the point in the HTML script – that is, relative to the other markup directives in the script – where the image is to be displayed using the tag

<IMG SRC = "*file name*">

where *file name* is the name of the file containing the image on the current page server. For example, assuming the file name containing the logo/crest of the UoW shown on the Web page in Figure 2.9 is *crest.jpeg*, this could be displayed on the page by replacing the first line in the body of the HTML script shown earlier in Figure 15.5(b) with

```
<IMG SRC = "crest.jpeg"> <H1> UNIVERSITY OF
WONDERLAND</H1>
```

Note, however, that some later versions of HTML may use the tag <OBJECT> to include an image where <OBJECT> applies not only to images but also to a number of other data items/objects. We shall expand on this later in Section 15.5.1.

Using the above method it is the browser that ultimately decides on the size and the position of the image. Hence it is possible that the heading will be displayed below the logo/crest with the above script. In addition, however, a number of optional attributes – also called parameters – can be defined with the IMG tag to inform the browser of the required position and other attributes. These include the ALT and the ALIGN attributes. The ALT attribute is used to specify a text string that should be displayed if the browser is not able to display an image. For example,

```
<IMG SRC = "crest.jpeg" ALT"UoW crest">
```

could be used to display the words *UoW crest* instead of the actual crest/logo if, for example, the user has disabled images or the related decompression software is not supported.

The ALIGN attribute is used to align an image with respect to either the display window or to some displayed text. In the first case, the related image can be displayed aligned to the LEFT window edge, which is the default, in the CENTER of the page, or aligned with the RIGHT edge. An example showing a segment of an HTML script to align an image in the center of the page is shown in Figure 15.7(a).

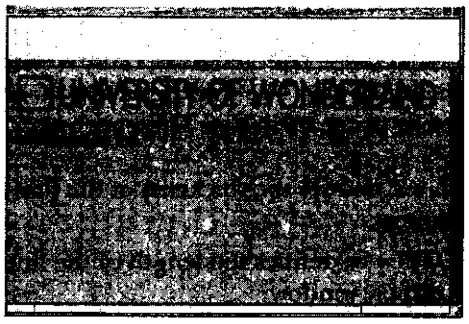
An image can also be aligned with respect to some given text. In this case the text following an IMG tag can start at various specified points relative to the image. For example, the text can start on the first line at the TOP of the image, the MIDDLE, or the BOTTOM. A segment of HTML script showing how the text starts at the middle of the image is shown in Figure 15.7(b).

Normally, before displaying any text that comes after a specified image, the browser waits until it receives the complete image to determine the amount of display space the image needs. For a large image this can delay the display of the complete page. To overcome this (and speed up the display process) the size of the image can be included with the image specification. Also, the size of the margin that should be left around the image. With this information the browser is able to reserve the requisite amount of display space and, while the image is being transferred and displayed, output any remaining text. The format used to do this is:

```
<IMG WIDTH = "x" HEIGHT = "y" HSPACE = "a" VSPACE = "b"
SRC = "image.gif">
```

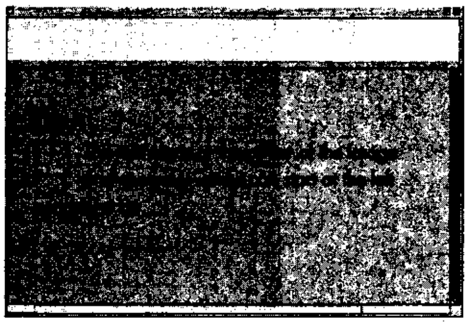
where x, y, a, and b are all specified in screen pixels. Note, however, that by specifying the dimensions in pixels the size of the displayed image will depend on the pixel resolution of the display. An image can also be used as a hyperlink by including the image specification within the hyperlink tags:

(a)



```
<BODY>
<IMG SRC = "crest.jpeg" ALIGN=LEFT><H1>UNIVERSITY OF WONDERLAND</H1>
<BR> Welcome to the - - -
  •
  •
  •
</BODY>
```

(b)



```
<BODY>
  •
  •
  •
<IMG SRC = "image.gif" ALIGN = MIDDLE> This text starts at the center of the image and
continues until it can start at the left window edge.
  •
  •
  •
</BODY>
```

Figure 15.7 Aligning in-line images: (a) with respect to the display screen; (b) with respect to subsequent text.

```
<A HREF = "http://www.UoW.edu/images"><IMG
SRC = "image.gif"></A>
```

The displayed image *image.gif* will then be displayed highlighted and the user can click on any part of this to activate the hyperlink.

Lines

A horizontal line – referred to as a rule – can be displayed from within a page using the <HR> tag. The thickness, length, and position of the line/rule can be varied by using attributes. These include:

- SIZE = s: defines the thickness of the line as a multiple of the default thickness;
- WIDTH = w: defines the length of the line as a percentage of the width of the display window;
- ALIGN = y: defines whether the line is aligned to the left, center, or right of the display window.

An example showing a selection of displayed lines and the related fragment of HTML script is given in Figure 15.8.

(a)

(b)

```
<BODY>
  •
  •
  •
<HR>
  •
  •
  •
<HR WIDTH = 50%  ALIGN = CENTER>
  •
  •
  •
<HR SIZE = 5 WIDTH = 75%  ALIGN = RIGHT>
  •
  •
  •
</BODY>
```

Figure 15.8 Horizontal lines: (a) a selection of displayed lines; (b) associated fragments of HTML.

15.3.5 Tables

Tables can be used in Web pages not only to display a particular set of data in tabular form but also to control the overall layout of a page. A table consists of one or more rows and one or more columns. The intersection of each row and column is called a **cell** and a cell can contain a string of text, a number, an image, a hyperlink or, if required, another table. Each column can have a heading and, if required, a heading can span multiple columns. A selection of the tags that can be used to create a table are shown in Figure 15.9(a), an example of a displayed table in part (b) of the figure, and the HTML script relating to this in part (c).

As we can see, both the headings and the contents of each cell are defined a row at a time starting at the left column using the <TH> and <TD> tags respectively. Note that, normally, the column headings are displayed in boldface by the browser and that the <CAPTION> tag is defined within the pair of <TABLE> tags. Also, as we can deduce from this example, an unboxed table can be used to control the layout of a page by dividing the page into regions/cells.

(a)

Opening tag	Closing tag	Use
<TABLE>	</TABLE>	Start and end of an unboxed table
<TABLE BORDER>	</TABLE>	Start and end of a boxed table
<TR>	</TR>	Start and end of a row
<TH>	</TH>	Start and end of a heading
<TD>	</TD>	Start and end of a cell content
<CAPTION>	</CAPTION>	Start and end of a table caption

(b)

NETWORK	CO/CLS	CBR/VBR
PSTN/ISDN	co	cbr
LAN	cls	vbr
ATM	co	vbr
Internet	cls	vbr

Network operating modes

(c)

```
<HTML><HEAD><TITLE> Example table</TITLE></HEAD>
<BODY>
<TABLE BORDER ALIGN = CENTER>
<TR><TH>NETWORK</TH><TH>CO/CLS</TH><TH>CBR/VBR</TH></TR>
<TR><TD>PSTN/ISDN</TD><TD> co </TD><TD> cbr </TD></TR>
<TR><TD>LAN</TD><TD> cls </TD><TD vbr </TD></TR>
<TR><TD>ATM</TD><TD> co </TD><TD vbr </TD></TR>
<TR><TD> Internet </TD><TD> cls </TD><TD> vbr </TD></TR>
<CAPTION> Network operating modes </CAPTION>
</TABLE>
</BODY></HTML>
```

Figure 15.9 HTML tables: (a) selection of tags; (b) an example of a displayed table; (c) HTML script for the table.

The position of the table on the display and the size of each cell is determined by the browser based on the maximum length of either the heading or the contents of each cell in a column and the number of rows. The contents of each heading and cell are then centered within the cell. Alternatively, the ALIGN attribute can be used with the <TABLE>, <TH>, and <TD> tags to align the table/heading/cell contents either to the left edge of the display/cell, the right edge, or the center. Two examples showing the format used are:

```
<TABLE BORDER ALIGN = CENTER>
<TH ALIGN = LEFT>
```

In addition, a user can define the size of the table themselves by specifying either the number of pixels to be used or as a percentage of the actual table size relative to the size of the display window. The format used is:

```
<TABLE BORDER WIDTH = 50% LENGTH = 50%>
```

Also, the size of individual cells can be defined by adding attributes to the <TH> and <TD> tags. Some examples are:

```
<TH ROWSPAN = 2> <!- -> the depth of the heading should be 2 rows - ->
<TD COLSPAN = 3> <!- -> the cell should span 3 columns - ->
```

15.3.6 Forms and CGI scripts

The previous subsections have been concerned with how a selection of the HTML directives/tags associated with text, images, and tables can be used to specify the contents of a Web page, and how the page is displayed on the screen of a client machine by the browser. However, as we saw in Section 5.4.2, in applications relating to e-commerce, for example, in addition to the server returning the contents of a file/page that have been requested by a browser, it is also a requirement for the server to receive and process information that has been input by the user. For example, payment card details and other information relating to the purchase of a ticket or product. As we saw, this is entered by means of a fill-in form and, typically, the entered information is then sent back to the server where it is processed by a piece of software called a common gateway interface (CGI) script. In this section we expand upon both these topics.

Forms

A form provides the means for the browser to get input from a user. A form is declared within an HTML page between the <FORM> and </FORM> tags. In addition, the <FORM> tag has two mandatory attributes, ACTION and METHOD. The ACTION attribute specifies the URL of the server where the data entered by the user should be sent and includes the path name and the name of the file containing the CGI script that will process the data. The METHOD attribute specifies how the entered data should be sent. When using HTTP it is always set to POST which means that the data will be sent using a POST request message over a previously established TCP connection between the browser and the named server. An example showing the format used is:

```
<FORM ACTION = "http://company.com/cgi-bin/orderform1"
METHOD = POST>
```

A form can contain a number of alternative ways for obtaining input from a user. These include fill-in boxes for textual input and check-boxes or pull-down menus for making selections. Then, when the user has completed filling in the form, the user can either initiate the sending of the entered information to the named server (if all is well) or, if not, reset the form to its initial state and start again.

When creating a form – normally within a page – to obtain input from a user, the <INPUT> tag is used. There is a range of attributes associated with the tag which determine how the information is to be obtained. A selection of these are as follows:

- <INPUT TYPE = TEXT NAME = "aname" Size = "n" >: this is used to create a fill-in box of length n characters. The entered text is given an identifier of the value in NAME, that is, *aname*;
- <TEXTAREA NAME = "aname" ROWS = "m" COLS = "n"> </TEXTAREA>: this is similar to the previous type except that the fill-in box comprises m rows each of length n characters. Normally, the box has a scroll bar for more than 2 rows;
- <INPUT TYPE = PASSWORD NAME = "aname" SIZE = "n">: this is similar to TEXT except that the entered password is displayed on the screen of the browser as a string of * characters, one per entered character;
- <INPUT TYPE = CHECKBOX NAME = "aname" VALUE = "avalue">: this is used to select a single option from a list of options of which the user can select more than one. All the options in the list have the same NAME ("aname") but each option has a unique VALUE;
- <INPUT TYPE = RADIO NAME = "aname" VALUE = "avalue">: this is the same as CHECKBOX except that the user can select only one from the list;
- <INPUT TYPE = SUBMIT VALUE = "aname">: this is used to create a submit button with the name given in the VALUE attribute displayed on the face of the button. When selected, this causes the browser to send the set of data entered by the user to the server;
- <INPUT TYPE = RESET VALUE = "aname">: this is used to create a reset button with the name given in the VALUE attribute displayed on the face

of the button. When selected, this causes the browser to reset the form to its initial state;

- <INPUT TYPE = BUTTON VALUE = "aname">: this is used to create additional buttons each of which can have a script associated with it that is invoked when the button is activated.

An example of a displayed form that has been created using a selection of the above is shown in Figure 15.10(a) and the HTML script associated with this is given in part (b). Note that if no TYPE is specified it is assumed that the INPUT is plain text.

Once the user activates the SUBMIT button – given the label *Submit request* in the example – this causes the browser to send the set of data entered by the user to the CGI script/program in the server named in the URL of the ACTION attribute. The data is sent in a POST request message containing the name of each variable followed by an = character and the value entered by the user. Note, however, that only those variables that have an entered value are sent. Each variable name and its associated value is separated by an & character and any spaces in the entered value are replaced by a + character. For example, assuming the displayed form shown in Figure 15.10(a), an entered set of data might comprise the block of characters:

```
name=FirstName+Surname&address=AStreet+ATown+ACountry
&phoneno=888-99999&ProdType=modems&ProdType=hubs&
PurchDate=now
```

with no spaces between the characters. Hence assuming the URL shown in part (b), the HTTP in the browser first establishes a TCP connection to port 80 – the HTTP well-known port – in server *www.NetCo.com*. The block of characters is then sent over this connection using a POST request message with a file name of */cgi-bin/literature* and

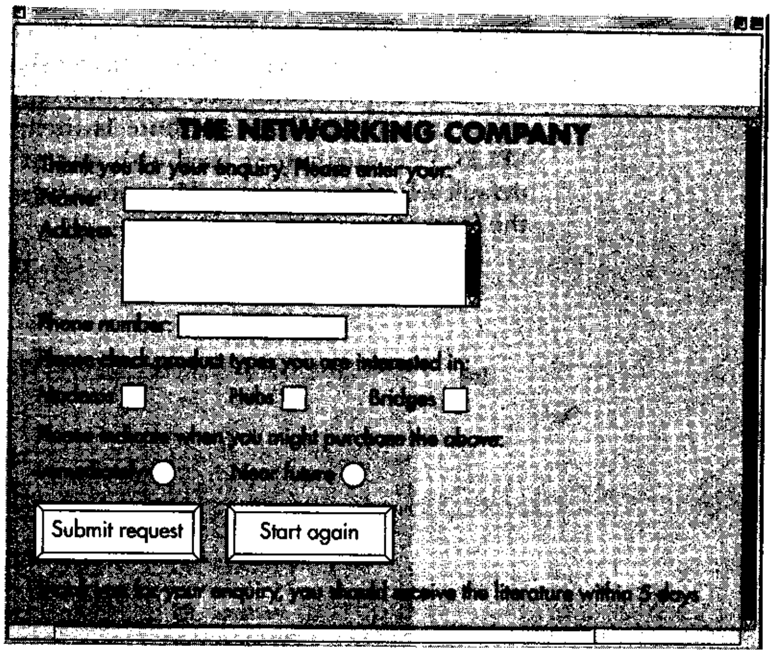
Content-Type: *text/html*

Content-Length: *115*

in the message header. This is followed by a blank line and the block of 115 characters in the message body.

Normally, on receipt of a POST request message with a directory name of *cgi-bin* the HTTP in the server invokes the CGI script/program in the named file and passes the contents of the request message to it for processing. The CGI script, after processing the contents of the POST message, may then return in the POST response message that it returns to the HTTP in the browser, a message such as that shown at the bottom of the display in the figure or, if some information is missing, a request to fill in the form again. Alternatively, it might return a Web page containing descriptions of the selected products.

(a)



THE NETWORKING COMPANY

Thank you for your enquiry. Please enter your:

Name:

Address:

Phone number:

Please check product types you are interested in:

Modems Hubs Bridges

Please indicate when you might purchase the above:

Immediately Near future

Submit request Start again

Thank you for your enquiry, you should receive the literature within 5 days

(b)

```
<HTML><HEAD><TITLE> Literature Request </TITLE></HEAD>
<BODY>
<H1> THE NETWORKING COMPANY </H1>
<FORM ACTION = "http://www.NetCo.com/cgi-bin/literature" METHOD = POST>
Thank you for your enquiry. Please enter your: <P>
Name: <INPUT NAME = "name" SIZE = 30> <P>
Address: <TEXTAREA NAME = "address" ROWS = 3 COLS = 40> </TEXTAREA>
Phone number: <INPUT NAME = "phoneno" SIZE = 20><P>
Please check product types you are interested in: <P>
Modems <INPUT TYPE = CHECKBOX NAME = "ProdType" VALUE = "modems">
Hubs <INPUT TYPE = CHECKBOX NAME = "ProdType" VALUE = "hubs">
Bridges <INPUT TYPE = CHECKBOX NAME = "ProdType" VALUE = "bridges"><P>
Please indicate when you might purchase the above: <P>
Immediately <INPUT TYPE = RADIO NAME = "PurchDate" VALUE = "now">
Near future <INPUT TYPE = RADIO NAME = "PurchDate" VALUE = "later"><P>
<INPUT TYPE = SUBMIT VALUE = "Submit Request">
<INPUT TYPE = RESET VALUE = "Start Again"><P>
</FORM></BODY></HTML>
```

Figure 15.10 HTML forms: (a) an example of a displayed form; (b) HTML script for the form.

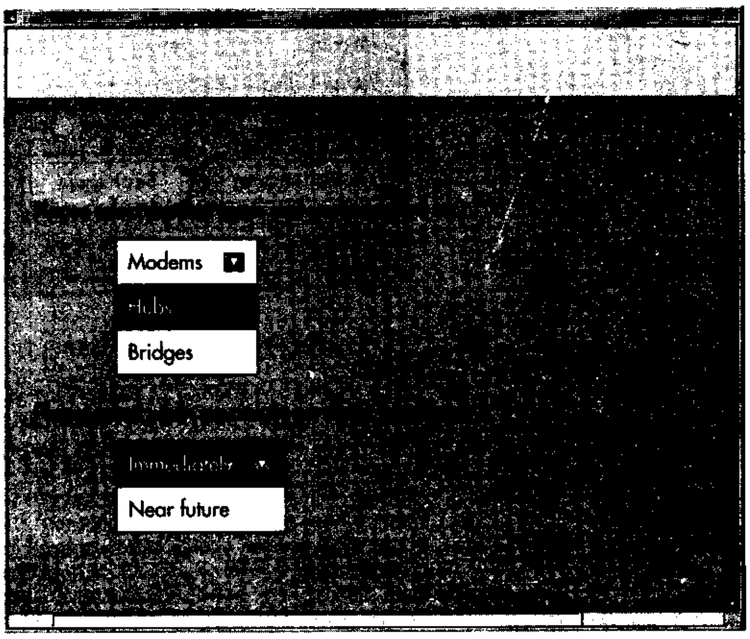
An alternative way of selecting an item from a list of options is in the form of a pull-down menu. A menu is created by defining each option between the <SELECT> and </SELECT> tags. Associated with the opening <SELECT> tag is a NAME attribute which is assigned the name of the list of options. Also, if more than one option can be selected, a MULTIPLE attribute. An example showing how the two list of options in the example in

Figure 15.10 could be displayed in the form of pull-down menus is shown in Figure 15.11(a) and the HTML script for this is given in part (b).

As we can see, since more than one option can be selected from the first menu, the MULTIPLE attribute is included. We can also see that the SELECTED attribute is used with one of the <OPTION> tags to show a default selection at start-up. The option selected with *PurchDate* is then sent in the form (say):

&PurchDate = Immediately

(a)



(b)

```
•
•
•
Please select product types you are interested in: <P>
<SELECT NAME = "ProdType" MULTIPLE>
<OPTION> "Modems"
<OPTION SELECTED> "Hubs"
<OPTION> "Bridges"
</SELECT><P>
Please select when you might purchase the above:
<SELECT NAME = "PurchDate">
<OPTION SELECTED> "Immediately"
<OPTION> "Nearfuture"
</SELECT><P>
•
•
•
```

Figure 15.11 HTML pull-down menus: (a) two examples; (b) HTML script.

15.3.7 Web mail

In addition to a user communicating with an email server – to send and receive mail messages – using an (email) user agent and, say, the POP3 protocol, it is also possible for a user to communicate with an (HTTP-enabled) email server using a Web browser and HTTP. In this case, the browser performs the user agent functions and all message transfers between the browser machine and the user's email server are carried out using HTTP rather than POP3.

As we saw in Section 14.3, a protocol like POP3 is used to transfer messages to/from the user agent and the user's email server over a point-to-point link. Using a browser and HTTP, however, has the advantage that, since any browser can be used to access a (registered) user's email server, the browser can be located anywhere around the world. The disadvantage is that accessing and sending mail in this way can be relatively slow. As we saw in the last section, since information – an entered email message in HTML for example – must be returned by the browser to the server, then all interactions with the email server must be through the intermediary of a form and an associated CGI script. In general, therefore, if a conventional email user agent can be used, this is the preferred choice when the user is working at his or her home or place of work. A Web browser is used when the user is away from home. In both cases, however, all message transfers between the email servers involved are carried out using SMTP.

15.3.8 Frames

Frames are used to enable the user to display and interact with more than one page on the display window of the browser at the same time. This is achieved by dividing the display window into multiple self-contained areas. Each area is called a **frame** and a separate page can be displayed in each frame. The user is able to interact with the page displayed in one frame while the pages in each of the other frames remain unchanged on the screen.

To divide the display window into multiple frames the start and end tags <FRAMESET> and </FRAMESET> are used. Associated with the <FRAMESET> tag are two attributes: COLS, which is used to divide the display vertically, and ROWS which is used to divide the display horizontally. For example, to divide the display vertically into two frames of equal size the following definition is used:

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

It is then possible to subdivide one or both of these frames using the ROWS attribute. For example, the left frame can be divided by following the previous definition with:

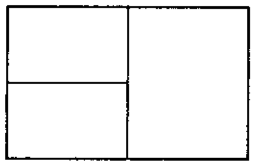
```
<FRAMESET ROWS = "70%,30%">
```

Once the display has been divided into the required number of frames (each of a defined size), the URL (or local file name) of the page to be displayed in each frame is defined using the SRC attribute in a <FRAME> tag. The HTML script of the page to be displayed in each frame is defined independently in the standard format (using all of the previously described features) and is stored in the related URL or local file name. Two example structures are shown in Figure 15.12(a) and the HTML script associated with each structure in part (b). Note that the <FRAMESET> tag effectively replaces the <BODY> tag when frames are being used.

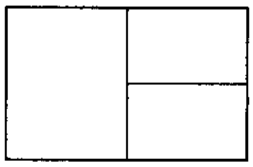
In these two examples, any images and/or hyperlinks in the displayed pages are accessed by the browser in the standard way and displayed in the frame displaying the related page. In addition, it is possible for a hyperlink in a page displayed in one frame to be used to access and display a page in one of the other frames. To do this it is necessary to give a name to the frame that is to be used to display the page when the frame is defined; for example, left, right, and so on. This is done using the NAME attribute with the <FRAME> tag. The same name is then added to the URL (or local file name) of the page that is to be displayed in this frame using the TARGET attribute. An example illustrating this feature is shown in Figure 15.13(a) with the HTML script in part (b) of the figure.

The example relates to that shown earlier in Figure 15.5 and the modifications that are necessary to the HTML script shown in Figure 15.5(b) to display the home page in the left frame and the selected pages from this in

(a) (i)



(ii)

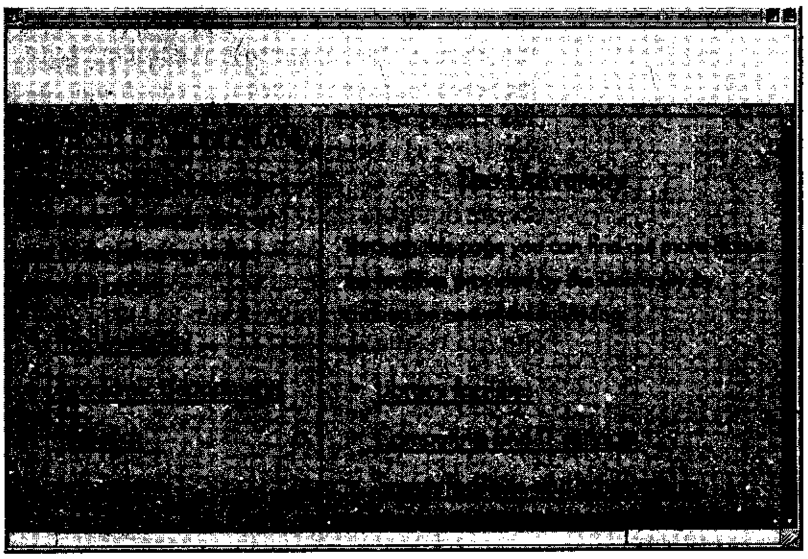


(b) (i)
```
<HTML><HEAD><TITLE>Frames1</TITLE></HEAD>
<FRAMESET COLS = "50%, 50%">
<FRAMESET ROWS = "50%, 50%">
<FRAME SRC = "URL/local file name of the page to be displayed in the top-left frame">
<FRAME SRC = "URL/local file name of the page to be displayed in the bottom-left frame">
<FRAME SRC = "URL/local file name of the page to be displayed in the right frame">
</FRAMESET></HTML>
```

(ii)
```
<HTML><HEAD><TITLE>Frames2</TITLE></HEAD>
<FRAMESET COLS = "50%, 50%">
<FRAME SRC = "URL/local file name of the page to be displayed in the left frame">
<FRAMESET ROWS = "50%, 50%">
<FRAME SRC = "URL/local file name of the page to be displayed in the top-right frame">
<FRAME SRC = "URL/local file name of the page to be displayed in the bottom-right frame">
</FRAMESET></HTML>
```

Figure 15.12 HTML frames: (a) two example frame divisions; (b) order of the related HTML scripts.

(a)



(b)

```
<HTML><HEAD><TITLE> Frames2 </TITLE></HEAD>
<FRAMESET COLS = "35%, 65%">
<FRAME SRC = "http://www.UoW.edu">
<FRAME SRC = "university" NAME = "right">
</FRAMESET></HTML>
```

(c)

```
         •
         •
         •
<UL><LI><A HREF = "university" TARGET = "right"> The University </A>
         •
         •
         •
    <LI><A HREF = "admissions" TARGET = "right"> Admissions </A>
         •
         •
         •
```

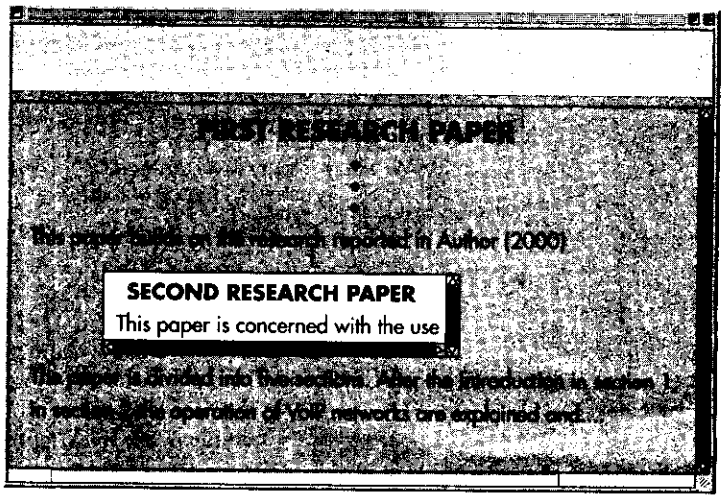
Figure 15.13 Nested frames: (a) example display composed of two vertical frames; (b) HTML script for the display; (c) modifications to the HTML script shown earlier in Figure 15.5(b).

the right frame are given in Figure 15.13(c). As we can see, for the frame on the right we have given it an initial URL of *The University* and a NAME = "*right*". Also, in order for the browser to know to display all the subsequently accessed pages in the right frame, each URL has an added attribute of TARGET = "*right*".

In addition to dividing the display window into multiple fixed-sized frames, it is also possible to display a second page in another frame that is defined in the HTML script of the currently displayed page. The second frame is called an **in-line frame** as it is created by inserting an <IFRAME> tag in the place in the current page where the frame is to be created. The <IFRAME> tag has a number of attributes which include SRC to specify the URL (or local file name) of the page to be displayed in the second frame, WIDTH and HEIGHT to define the size of the second frame, and FRAMEBORDER to indicate whether the frame should have a border (=1) or not (=0). An example is shown in Figure 15.14(a) and the HTML script for this in part (b).

As we can see, in this example the in-line frame is used to display the contents of a second paper that is referenced in the first paper. In this way, if the reader of the first paper is not familiar with the referenced paper then he or she can read it through the in-line frame.

(a)



FIRST RESEARCH PAPER

•
•
•

This paper builds on the research reported in Author (2000)

SECOND RESEARCH PAPER
This paper is concerned with the use

The paper is divided into five sections. After the introduction in section 1, in section 2 the operation of VoIP networks are explained and...

(b)

```
<HTML><HEAD><TITLE> In-line frames </TITLE></HEAD>
<BODY>
<H1> FIRST RESEARCH PAPER </H1><P>
        •
        •
        •
This paper builds on the research reported in Author (2000) <P>
<IFRAME SRC = " URL of page Author (2000)" WIDTH = "600" HEIGHT = "200"
      FRAMEBORDER = "1"></IFRAME><P>
The paper is divided into five sections. After the introduction in section 1, in section 2 the
operation of VoIP networks are explained and...
        •
        •
        •
```

Figure 15.14 In-line frames: (a) a segment of a displayed page containing an in-line frame; (b) a segment of the HTML script to produce this.

15.4 Audio and video

With Web pages comprising text and/or images, the contents of the file containing the page are downloaded from the server to the client machine. The browser then displays the page contents on the screen and, in the case of a large file/page, the user can use the scrolling facilities to view the whole page.

In the case of audio and video, however, the volume of information to be transferred increases linearly with time and hence is determined by the duration of the audio/video clip. As we saw in Chapter 4, typical bit rates for compressed audio – for example MPEG layer 3 (**MP3**) – are 128 kbps for two-channel stereo and 1.5 Mbps for MPEG-1 video with sound. Hence even a short audio/video clip can require a significant time to download. For example, a 5 minute audio clip/track compressed using MP3 produces $5 \times 60 \times 128 \times 10^3$ bits or 38.4 Mbits. Even with a relatively high access rate of, say, 1 Mbps this requires 38.4 s to download which, for this type of application, may not be acceptable. For video, of course, the delay would be an order of magnitude larger.

Hence when a user requests a file containing (compressed) audio and/or video, except for relatively short files containing spoken messages and short audio and/or video clips, the contents of the file must be played out as they are being received. As we saw in Section 1.5.1, this mode of working is called **streaming**. Also, as we saw in Section 1.5.6, to overcome variations in the time between each received packet in the stream – jitter – normally a **playout buffer** is used. This operates using a first-in first-out (FIFO) discipline and typically, holds several seconds of audio and/or video. The received compressed bitstream is then passed through the buffer and output from the buffer does not start until the buffer is, say, half full.

Each of these functions is in addition, of course, to the decompression of the audio and/or video bitstream. To perform these various functions the browser uses a range of helper applications. Normally, these are referred to as **media players** as they form the interface between the incoming compressed media bitstream and the related sound and/or video output card(s). In the case of audio, the appropriate audio media player – MP3 for example – decompresses the audio bitstream taken from the playout buffer and passes it to the sound card. The latter first converts the decompressed bitstream(s) back into an analog signal(s) and, after amplification, the signal(s) is/are output to the speakers. For video, the browser first creates a window in the Web page from where the request was initiated and then passes the coordinates of the window to the selected video media player. The latter first initializes the video card with the assigned coordinates and, as it decompresses the video bitstream taken from the playout buffer, it passes it to the video card for rendering on the browser screen.

In addition, as we saw in Section 1.4.3, with entertainment applications such as audio/video-on-demand, the user requires control of the playout process using features such as pause and rewind. Hence with this type of application it is necessary for the media player to pass the control commands to the server. To do this, the media player is divided into two parts: the first that performs the preceding playout functions – playout buffering and decompression – and the second that manages the portion of the browser display window that has been assigned for the various control buttons. This displays and monitors the buttons on the screen and, when a button is selected, it first adapts the playout process – for example stops output if the pause button is activated – and then passes an appropriate command to a remote server. A protocol has been defined to perform this function called the **real-time streaming protocol (RTSP)**. The server then implements the command by, for example, stopping further output from the file.

Although it has been implied that the server is a conventional Web server, in most entertainment applications involving streaming, in order to meet the very high playout rates that are required when a large number of browsers are accessing the server simultaneously, special servers called **streaming servers** have to be used. In this section we shall expand upon several of these issues.

15.4.1 Streaming using a Web server

Before describing how an audio and/or video file is accessed using a streaming server, it will be helpful first to review how such files are accessed using a conventional Web server. Figure 15.15 shows the protocols that are used.

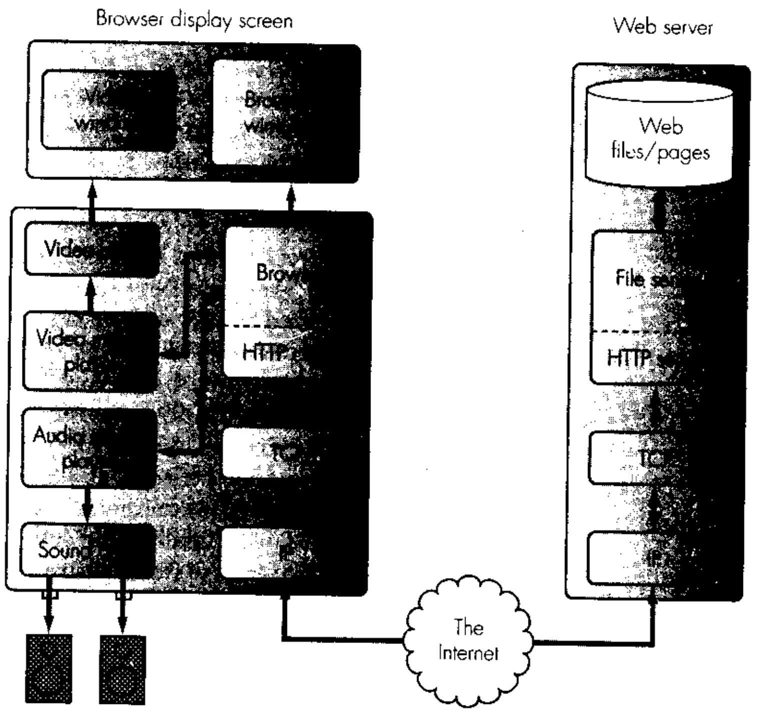


Figure 15.15 Schematic of audio/video streaming with a conventional server.

Using this structure, when the user clicks on a hyperlink in a page for an audio or video file, the browser follows the same procedure as for a text/image file. Hence the HTTP in the browser first establishes a TCP connection with the HTTP in the server named in the hyperlink. It then sends a request for the contents of the file named in the hyperlink using a GET request message. The server responds by returning the contents of the file in a GET response message. On receipt of this, the browser determines from the *Content-Type* field at the head of the message – for example *Content-Type = Audio/MP3* – that the accessed file contains audio that has been compressed using MP3. Hence the browser proceeds to invoke the MP3 media player and, at the same time, passes the contents of the compressed file to it. The media player then proceeds to decompress the contents of the file and outputs the resulting byte stream to the sound card.

As we can deduce from this, the disadvantage of this approach is that since the browser must first receive the contents of the file in its entirety, an unacceptably long delay is introduced if the contents are of any significant size. Hence for larger files, an alternative approach is used which enables the file to be sent directly to the media player rather than through the browser. A schematic diagram showing how this is achieved is shown in Figure 15.16.

Using this approach, when an audio and/or video vile is created, a second file is also created. The second file contains the URL of the first/original file – containing the compressed audio/video – and also a specification of the content type that is in the file. The second file is called the **meta file** of the original file or, because of its function, a **presentation description file**.

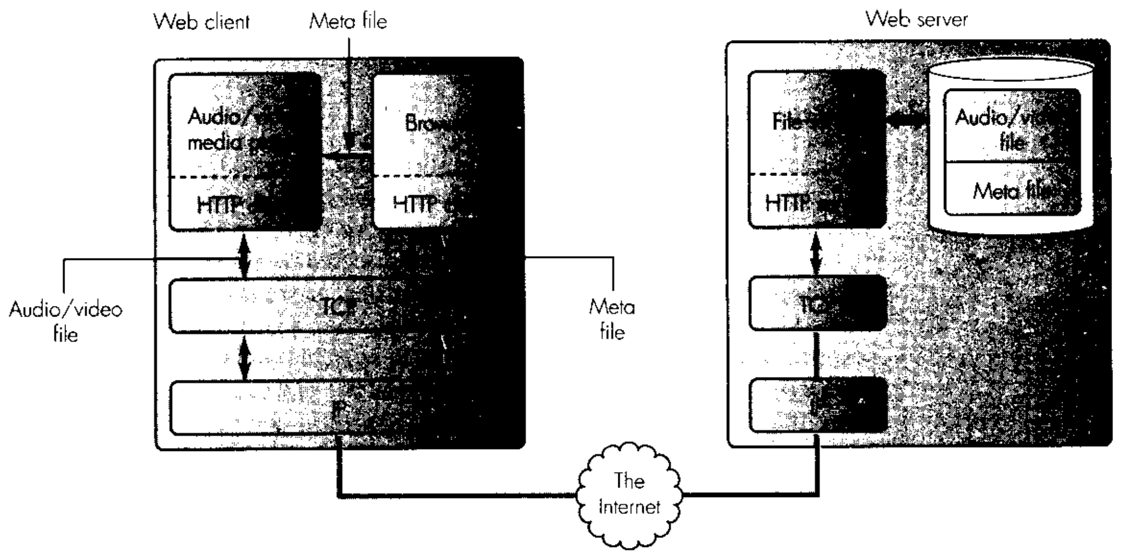


Figure 15.16 Audio/video streaming with a conventional server and a meta file.

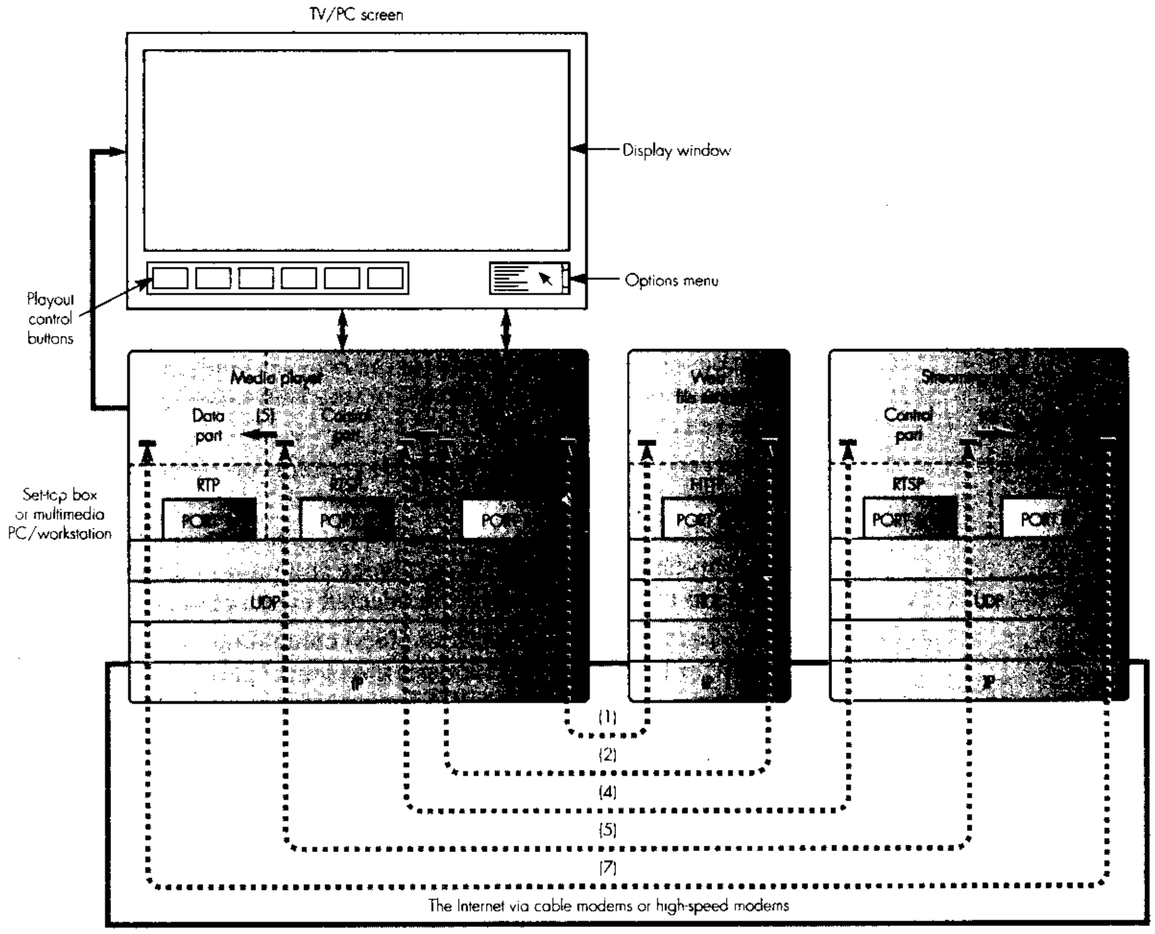
This also has a URL associated with it and, when the creator of a page wishes to include a hyperlink to an audio/video file in the page, he or she uses the URL of the meta file rather than that of the original file.

Thus when a user clicks on the hyperlink, the GET response message contains the contents of the meta file. The browser first accesses the *Content-Type:* field from it and then uses this to invoke the related media player as before but this time it simply passes the presentation description in the meta file to the media player. The media player, on determining that this is a meta file, reads the URL of the original file and then proceeds to obtain the contents of the original file in the normal way using HTTP/TCP. On receipt of the file contents, the media player simply streams the received compressed contents into the playout buffer. After a predefined delay to allow the playout buffer to partially fill, it starts to read the stream from the buffer and, after decompression, outputs the resulting byte stream to the sound/video card.

As we can see from the above, this approach removes the delays that are introduced when the file contents are accessed through the browser and hence it is widely used when the audio/video is stored on a conventional Web server. The limiting factor with this approach is that since the audio/video file is accessed in the same way as a text or image file using HTTP and TCP, as we saw in Section 12.3, TCP will transfer the file contents in segments and, if it detects a segment is missing, the TCP at the server side will retransmit it. In general, for files containing real-time information such as audio and/or video, the delays introduced by the TCP retransmission process mean that large playout buffers are required in the media player to mask the effect of a missing segment from the user. Because of this, the preferred transport protocol for audio and/or video files is UDP rather than TCP. This means that HTTP cannot be used and so a different file server from the Web server must be used to hold the audio and/or video files. This is called the streaming server.

15.4.2 Streaming servers and RTSP

As we indicated earlier in this section, the main demand for streaming servers is in entertainment applications such as audio-on-demand and movie/video-on-demand. Typically these are provided by either a TV or a multimedia PC/workstation via a set-top box that is connected to either a cable modem or a high-speed modem. With such applications, as we saw in Section 1.4.3, in order to meet the very high playout rates that are required when a large number of concurrent users are involved, special-purpose streaming servers are used. Also, as we indicated at the end of the last section, in order to overcome the delays introduced by the retransmission procedure associated with TCP, the preferred transport protocol is UDP. Hence when accessing an audio/video file from a streaming server, normally UDP is used with the real time transport protocol (RTP) to transfer the integrated audio and video. A typical set-up for movie/video-on-demand is shown in Figure 15.17. A similar set-up is used for audio-on-demand except no video is involved.



RSTP = Real-time streaming server Ports X/Y/B are ephemeral ports
Ports 80/554 are well-known ports of HTTP/RTSP servers respectively
Port A is allocated by the control part of the media player

Figure 15.17 Protocols associated with audio/video streaming.

As we can see, the set-up is similar to that shown in Figure 15.16 except the streaming server is separate from the Web server. Also, at the browser side, the media player is divided into two parts: a data part and a control part both of which interact with peer parts in the streaming server. The data part is concerned with the transfer of the integrated audio and video packet streams – see Section 5.5.1 – from the server to the client, the buffering of the incoming packet stream in the playout buffer, the demultiplexing and decompression of the audio and video components, and the output of synchronized audio and video to the respective media cards. The control part manages the playout process according to the commands entered by the user via the set of on-screen control buttons.

Typically, the screen of the TV/PC is divided by the browser into three windows. The first is for use by the browser itself to display a menu of audios (CDs) and movies/videos, the second for use by the control part of the media player to display the set of control buttons, and the third for use by the data part of the media player to display the video output. Also, as with streaming using a conventional Web server, associated with each file containing the integrated audio/video packet stream is a meta file containing the URL of the file and a description of such things as the compression algorithms used and the presentation format.

The sequence of steps that occur when the user selects a movie/video from the menu are identified as (1) through (7) in Figure 15.17. These are:

1. When the user clicks on a movie/video, the browser sends an HTTP GET request message for the related meta file to the Web server named in the URL of the selected hyperlink.
2. The Web server responds by returning the contents of the meta file to the browser in a (HTTP) GET response message.
3. The browser determines from the *Content-Type:* field in the meta file the media player – helper application – to invoke and passes the contents of the meta file to the control part of the selected media player.
4. The control part reads the URL of the file(s) containing the integrated audio/video packet stream and requests the control part in the streaming server (named in the URL) for permission to start a new session by sending an *RTSP SETUP* request message. Associated with this are a number of parameters including the name of the file containing the movie/video, the RTSP session number, the required operational mode (PLAY, although RECORD is supported also), the RTP port number to use, and authorization information. In response, if these are acceptable, the control part in the server returns an RTSP accept response message which includes a unique session identifier allocated by the server for use with subsequent messages relating to the session.
5. When the user clicks on the *play* button, the control part of the media player sends a *RTSP PLAY* request message – which includes the allocated session identifier – to the control part in the streaming server. The latter knows from the identifier that access permission has been granted and returns an acknowledgment to this effect to the control part in the media player. On receipt of this, the latter prepares the data part to receive the incoming integrated audio/video packet stream.
6. At the streaming side, after a short delay to allow the client side to prepare to receive the packet stream, the control part initiates the access and transmission of the packet stream using the allocated port number of RTP – port A – in the header of each UDP datagram.
7. The stream of packets containing the integrated audio and video are first passed into the playout buffer of the data part of the media player and, after a preset time delay, the packets are read from the buffer. Each packet in the stream is first identified – audio or video – and then decompressed using the previously agreed algorithm.

Once the play button has been selected and the movie/video started, the user may wish to activate further control buttons such as pause, visual fast forward, rewind, stop, and quit. In order to relay the appropriate command to the streaming server, RTSP has a corresponding set of control messages. A selection of these are shown in the example in Figure 15.18(a) and the format of some of the messages in part (b).

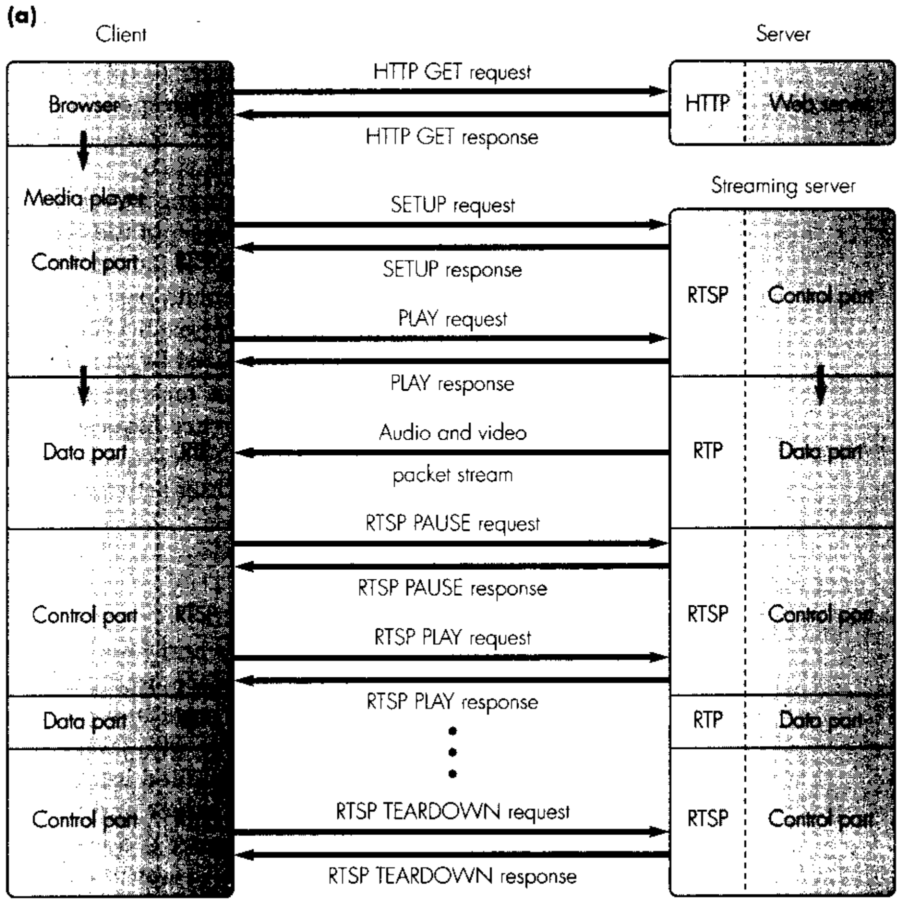
Note that in order for the streaming server to send the integrated stream of audio/video packets to the data part of the media player, the port number that has been allocated by the control part to RTP – port A – is included in the SETUP request message. The data part of the streaming server inserts this in the destination port field of each UDP datagram header. Note also that the session identifier – allocated by the control part in the streaming server – is returned in the response message to the SETUP request and that this is used subsequently in all further request messages relating to this session. Normally, the RTSP TEARDOWN request message is sent by the control part of the media server when the user activates the quit/end button.

15.5 Java and JavaScript

A short introduction to both Java and JavaScript was presented earlier in Section 5.4.4 and, as we saw, they are used primarily to add some form of action and interactivity to a Web page in a more flexible way than helper applications and plug-ins. Essentially, with Java, a program written in the Java programming language is first compiled into what is called an *applet*. Each applet is held in a file on a Web server and can be called from within an HTML page. When called, the applet is downloaded from the file (in the server named in the URL) in a similar way to downloading the contents of an image file. When the browser receives the applet code, however, it passes it directly to an integral piece of software called a *Java interpreter* which proceeds to execute the applet code. A simple example that shows how action can be added to a Web page is an applet which obtains an audio file and plays this out as background music while the page is displayed. Alternatively, a moving object/image could be displayed in a small window. A more complex example that shows how interactivity can be introduced is an applet which obtains and displays a digitized map on the browser screen and, given a pair of coordinates – for example entered by the user in a frame in a separate window – calculates and displays the best route between the two coordinates. Games playing within a Web page is implemented in a similar way.

Similar functionality can be obtained using JavaScript except, like HTML, this is a scripting language and the action/interactivity is obtained by embedding individual scripts written in JavaScript into an HTML page when it is written. Each script is entered into the HTML script of a page between a pair of tags and, when the start tag is encountered, the HTML interpreter invokes the JavaScript interpreter to execute the script.

(a)



(b)

Assume URL of movie/video meta file = http://www.movie server.com/movies/amovie.mfile
URL of movie/video file = rtsp://movieserver.com/movies/amovie.mpeg

SETUP Request:	SETUP/movies/amovie.mpeg RTSP/1.0 Accept: audio/MP3, video/MPEG1 Transport: RTP/UDP, port = A, Mode = PLAY
SETUP Response:	RTSP/1.0 200 Accepted Server: Movie player Location: movieserver.com Session: 1234
PLAY Request:	PLAY/movies/amovie.mpeg RTSP/1.0 Session 1234
PLAY Response:	RTSP/1.0 200 Accepted

Figure 15.18 Real-time streaming protocol (RTSP): (a) example message exchange sequence; (b) a selection of message formats.

Both Java and JavaScript have many similar features to those provided by the C and C++ programming languages. Hence a complete description of them is outside of the scope of this book. In this section, therefore, we restrict our discussion to how an applet/script written in these languages is incorporated into an HTML page.

15.5.1 Java

Java applets provide a browser with similar functionality to helper applications and plug-ins. In the case of the latter, however, they are an integral part of the browser whereas a Java applet is downloaded from a Web server when it is required. The advantage of this is that the applet code can be changed at any time without modifying the browser code. Then, when the old version of the applet is replaced with the new version, this is downloaded automatically the next time the applet is called. For example, if the applet implements a compression algorithm and a new one is developed, the existing applet can be replaced by a new applet without changing the page contents or the browser.

In order for the downloaded code of an applet – the file of which has the type **class** – to run on a range of different computers/machines, when an applet is written it is compiled into a machine-independent code called **bytecode** and it is this version of the applet that is stored in the file on the Web server. Essentially, the bytecode of an applet is an intermediate code between the high-level Java programming language code and the machine-code version of the applet produced for running directly on a specific target machine. To run the bytecode version of an applet, a browser that supports applets has an integral piece of software – or sometimes a helper application – called a **bytecode interpreter**. This parses the bytecode to identify the individual commands – called **methods** – that it contains and then executes each of these using a related procedure/function written in the machine code of the target machine. Hence when an applet is called from within a Web page, the downloaded bytecode is simply passed by the browser to the bytecode interpreter for execution.

Applet tags

In versions of HTML before HTML 4.0, the inclusion of a Java applet in an HTML page is carried out using the <APPLET> tag. For example, assuming the URL of the current Web page is:

http://www.UoW.edu/java-apps/"

to include the applet stored in the file *bgsound.class* on the same server as the current Web page, the script

<APPLET CODE = "*bgsound.class*"></APPLET>

is used. When the HTML interpreter in the browser encounters the <APPLET> tag in the HTML code, it proceeds to obtain the contents of the file – the bytecode – from *bgsound.class* and passes it to the bytecode interpreter. The latter proceeds to interpret the bytecode of the applet which, typically, accesses a specific file of compressed audio, decompresses it and then outputs the resulting byte stream to the sound card to play the background music.

As we can deduce from this, there is no separation between the applet bytecode and the data on which it operates. Also, there is no type definition associated with the data. So to obtain more flexibility, in HTML 4.0 and later versions, the <APPLET> tag is replaced with the more general <OBJECT> tag. Using this, in addition to Java applets, a number of other types of object can be included within an HTML page. These include an image (file), an audio and/or video (file) or, if required, another Web page (file). To achieve this added flexibility, the <OBJECT> tag has a number of attributes associated with it. These include:

- CODEBASE: this is the URL of the file server and the pathname of where the objects – for example the applet and any data it operates on – are located;
- CLASSID: this specifies the name of the file containing the agent – for example the applet bytecode – that will render the data. Normally, the file name of an applet is prefaced by the new URL type *java*;
- DATA: this specifies the name of the file containing the object/data on which the agent specified in CLASSID operates;
- CODETYPE: this specifies the type of the object/data defined in DATA in MIME format;
- ALIGN, HEIGHT, WIDTH, ALT: these have the same meaning as those we defined earlier in Section 15.3.4 relating to images.

An example declaration showing the use of some of these attributes – using the same URL as before – is as follows:

```
<OBJECT CODEBASE = "http://www.UoW.edu/java-apps/"
        CLASSID = "java:bgaudio.class"
        DATA = "bgaudio.data"
        CODETYPE = "audio/MP3"> </OBJECT>
```

In this example when the HTML interpreter in the browser encounters the <OBJECT> tag, it requests the Java applet stored in the file specified in CLASSID to be downloaded by the server specified in CODEBASE (which also includes the pathname of the file). The name of the file specified in DATA and the datatype – audio/MP3 – specified in CODETYPE are then passed to the bytecode interpreter together with the bytecode of the applet. In the case of an image or video object, the HEIGHT and WIDTH attributes would be used to specify the size of the window that should be used to display the image/video.

Java basics

Java is an object-oriented language which means that almost everything is defined in the form of an **object**. Normally, like a procedure or function in a programming language like Pascal or C, an object contains one or more variables encapsulated within it. Also, associated with each object are one or more operations called **methods** and it is these that are invoked to manipulate the variables within the object. This concept is referred to as **encapsulation**.

Multiple instances of an object can be created – either statically or dynamically during the execution of a program/applet – each of which is said to be an instance of the same object *class*. A typical Java applet comprises many objects – and hence methods – each of which is an instance of a particular object class. Also, in addition to writing your own object class definitions from scratch, like all programming languages, there is a large library of standard object classes that can be included with your own. These are grouped into what are called **packages** and two examples are:

- Java.io: all input and output such as reading a file, outputting a byte stream, and so on is done using the methods associated with the object classes in this package;
- Java.applet: this contains object classes and methods for getting a Web page from a given URL, displaying a Web page, decompressing and playing out the contents of an audio and/or video file, and so on.

There is also a range of Java development kits available that can be used to create applets. A good source of information for this is at

http://www.javasoft.com/

15.5.2 JavaScript

As we indicated earlier, despite its name, JavaScript is a completely different language from Java. It is a scripting language and the script is embedded within an HTML Web page between the <SCRIPT> and </SCRIPT> tags. Providing the browser is able to interpret JavaScript code, when it detects the <SCRIPT> tag it proceeds to interpret the code up to the </SCRIPT> tag as JavaScript rather than HTML.

Unlike HTML, JavaScript has many high-level programming language features similar to those available with C and C++. For example, a variable can be of type boolean, numeric, or string. Also, arithmetic, logical, and bitwise operators are supported as are for( ) and while( ) control loops. Functions are also supported.

Objects

JavaScript is object-oriented. Each object has one or more methods associated with it that allow the object to be manipulated in some way. For example, the current displayed Web page – referred to as a document in JavaScript – is an example of a library/predefined object class and one or more expressions (such as a text string) can be displayed in the page window using

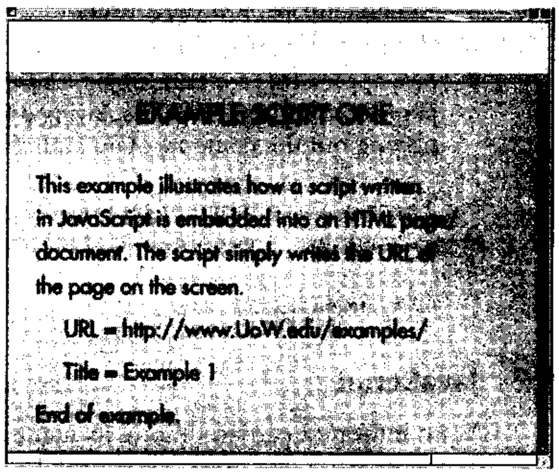
document.write (text string/expression(s))

Here the object name is *document* and the method is *write* – *writeln* can also be used. Additional methods are also provided to determine such things as:

- the URL of the document: *document.URL*,
- the title of the document: *document.title*.

A simple example that shows how a JavaScript is embedded into an HTML page and how the above two methods can be used is shown in Figure 15.19.

(a)



EXAMPLE SCRIPT ONE

This example illustrates how a script written
in JavaScript is embedded into an HTML page/
document. The script simply writes the URL of
the page on the screen.

URL = http://www.UoW.edu/examples/

Title = Example 1

End of example.

(b)

```
<HTML><TITLE> Example 1 </TITLE>
<BODY>
<H1> EXAMPLE SCRIPT ONE </H1>
This example illustrates how a script written in JavaScript is embedded
into an HTML page/document. The script simply writes the URL of the page
on the screen: <P>
<SCRIPT LANGUAGE = "JavaScript">
document.write ("URL =", document.URL) <P>
document.write ("Title =", document.title) </SCRIPT>
<P> End of example.
</BODY></HTML>
```

Figure 15.19 JavaScript principles: (a) an example of a displayed page/document containing a JavaScript embedded into it; (b) the script for the page.

Another widely used predefined object class that is used to obtain interactivity is *window* and again several methods are provided with this. For example, to create a new window – as a dialog box for example – the *open* method is used. An example statement is:

"MyWindow"= window.open (width = "200", height = "200", scrollbars = "Yes")

This would open a new window called *MyWindow* of a defined size (in pixels) and with scrollbars. Some text could then be included in the script and this would be displayed within the window. The same window can later be closed using the *close* method. The statement in this case would be:

Mywindow.close

Other widely used methods associated with the window object class are:

- *window.confirm ("message")*: this is used to display a confirmatory dialog box with the specified message within it and an OK and Cancel button;
- *window.prompt ("message")*: this is used to display a prompt dialog box with the specified message within it and an input field;
- *window.alert ("message")*: this is used to display a box with the specified message within it.

Some other predefined object classes are array, boolean, date, and math. For example, the math object class includes methods to return a value for pi and to carry out the sin and cos functions. An example segment of script showing their use is:

```
avariable = math.PI*math.cos (math.PI/6)
```

Alternatively, when the script contains several occurrences of methods from the same object class the *with* statement can be used:

```
with (math)
{avariable = PI*cos (PI/6)}  //comment: note the use of curly braces
```

A new instance of an object class can be created within a script using the *new* operator and the keyword *this* is used to refer to the current object in which the keyword appears.

Forms and event handling

We introduced the <FORM> and <INPUT> tags earlier in Section 15.3.6. As we saw, these provide the means for the browser to get input from a user which, typically, is sent to a CGI script in a remote server for processing. In addition, a form can be declared without the ACTION and METHOD attributes and the input processed locally by the browser using a JavaScript script. Also, in order to enhance this capability, JavaScript allows a number of what are called **event handlers** to be specified. Then, when a specific event occurs – for example the user clicks on an entry in a displayed table of values – a related block of JavaScript code is invoked which, for example, might be to perform a computation on the value that has been clicked on. Two examples of event handlers that are supported are:

- onBlur: a Blur event occurs when an option from a list on a form is selected or some text is entered in a text field;
- onClick: the Click event occurs when an option on a form is clicked. The option can be selected using either a button, check-box, radio, reset, submit, or link.

An example showing a segment of script that illustrates the use of an event handler is given in Figure 15.20. In this example the user is prompted for his or her user name and password and, when each is entered, the related onBlur event handler is invoked to check its validity. As we can see, each is checked using a different function and, if either fails, an appropriate message is output in an alert window.

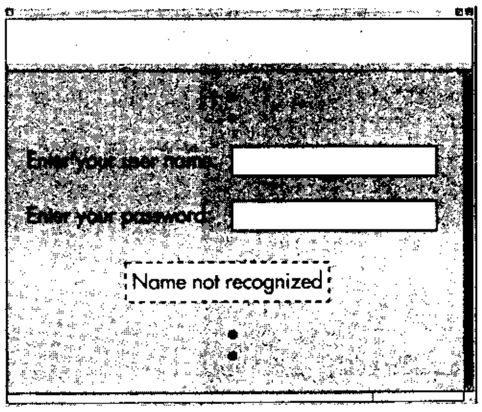
15.6 Security

When carrying out a transaction over the Web relating to an e-commerce application, since in many instances this involves sending details of a user's payment card, the security of such transfers is vitally important. For example, an eavesdropper with knowledge of Internet protocols could readily intercept the information entered on the order form and, having got the name and other details about the card, proceed to use these to carry out purchases of their own. A second potential pitfall is that the Web site from where the purchase is being made may not in reality have anything for sale and, prior to the agreed delivery date, abscond with the money that it has collected. In addition, in electronic banking (e-banking) and other financial applications, a client may masquerade as another person. Any security scheme, therefore, must be able to counter each of these threats.

15.6.1 SSL

An example of a widely used scheme is the **secure socket layer (SSL)** protocol. As the name implies, SSL operates at the socket interface which, as we saw earlier in Section 12.3.1, is between the transport layer (TCP) and the application layer in the TCP/IP reference model. Essentially, SSL carries out the authentication of the server by the client – and the authentication of the

(a)



(b)

```
•
•
•
<FORM>
Enter your user name: <INPUT NAME = "username" onBlur = "checkname (username.value)">
Enter your password: <INPUT NAME = "password" onBlur = "checkpassword (password.value)">
</FORM>
•
•
•
<SCRIPT>
function checkname (aname)
{if(aname! = "myname") window.alert ("Name not recognized")}
function check password (apassword)
{if (apassword! = "mypassword") window.alert ("Wrong password")}
</SCRIPT>
```

Figure 15.20 Example showing event handling within a form: (a) a portion of the display; (b) the script associated with it.

client by the server if required – by using a recognized certification authority – see Section 13.6 – and the establishment of a symmetric encryption algorithm and key for the session. It then uses the key – called a **session key** – to encrypt/decrypt all of the messages that are transferred as part of the transaction. The location of the SSL protocol in the stack is shown in Figure 15.21(a) and a summary of the steps that are followed to establish a secure socket connection are listed in part (b) of the figure.

As we can see, the HTML interpreter part of the browser has two pathways/sockets available to it, an insecure socket connection and a secure socket connection. When a user clicks on a link to an SSL-enabled server, the protocol/method part of the URL is *https:* rather than *http:* . On detecting this, the HTML interpreter invokes the SSL protocol code which proceeds to carry out the steps shown in Figure 15.21(b).

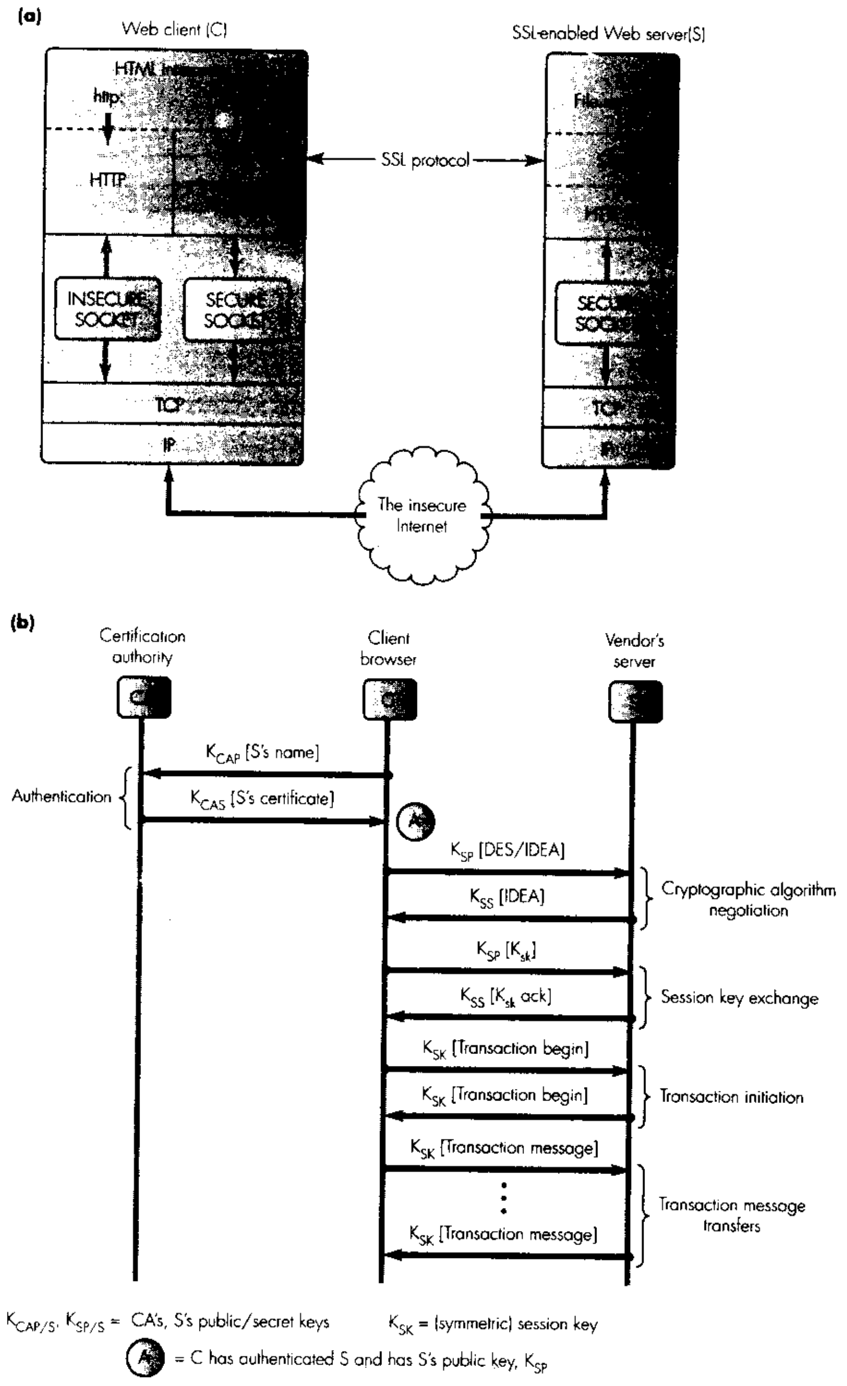


Figure 15.21 The secure socket layer (SSL) protocol: (a) protocol stack; (b) outline of the authentication and transaction initiation phases.

As we can deduce from the list of steps, the establishment of a secure socket connection is carried out by the exchange of an ordered set of HTTP request/response messages which collectively form the SSL protocol. When interpreting the steps shown in the figure, the following should be noted:

- **Authentication using a CA:**
  - C authenticates S by first checking that the CA named in S's Web page is on the list of recognized CAs. If so, C reads the public key of the CA from the list and then requests S's certificate from the CA by sending it S's name in a request message. The CA then sends the certificate in a response message.
  - On receipt of the certificate, C decrypts it using CA's public key and checks that the name on the certificate is that of S. If so, C assumes S is authentic and reads the public key of S from the certificate.
- **Cryptographic algorithm negotiation**:
  - Once S has been authenticated, C proceeds to negotiate with S a suitable (symmetric) cryptographic algorithm for the transaction.
  - To do this, C sends its preferences – DES or IDEA for example – together with proposed operational parameters in a request message to S. S responds with its choice of one of these in the response message.
- **Session key exchange**:
  - C generates a random (symmetric) key for the transaction and encrypts it using S's public key. C then sends the encrypted key in a message to S.
  - On receipt of the message, S decrypts the message using its own private key and returns a response message to C acknowledging that it, too, now has the session key.
- **Transaction initiation:**
  - C sends an encrypted message to S informing S that it is now ready to start the transaction and that all future messages will be encrypted.
  - On receipt of this, S returns an encrypted response message that it, too, is now ready to start the transaction.
- **Transaction information transfer**:
  - Once a secure socket connection has been established, each message relating to the transaction that is sent/received over the socket is encrypted/decrypted using the agreed symmetric session key.

When using SSL in banking and other financial applications, normally, when a client starts a new session, it is the SSL in the server that authenticates the client before entering into a transaction.

15.6.2 SET

A potential loophole with SSL is that although the scheme allows the client to authenticate that the server is a recognized server, because the server's bank is not involved in the authentication process, the server's certificate does not guarantee that it is authorized to enter into transactions that involve payment cards. Similarly, the client's certificate does not guarantee that the client is using his or her own card. To counter these loopholes, the major card companies have introduced a scheme that has been designed specifically for card transactions over the Web/Internet. The scheme is called **secure electronic transactions** (SET).

In the SET scheme, in addition to the client (the purchaser) and server (the vendor), the client and vendor's banks are also involved also in carrying out a transaction. The purchaser, vendor, and vendor's bank all have certificates and, in the case of the purchaser and vendor, the certification authority is their respective bank. The purchaser's certificate, in addition to the purchaser's public key, also contains the name of the purchaser's bank. This enables, firstly, the purchaser to verify that the vendor is allowed to enter into payment card transactions and secondly, that the purchaser is using a legitimate card.

In practice, the SET scheme is relatively complex. However, an outline of the sequence of steps followed to carry out a card purchase electronically is shown in Figure 15.22. The following should be noted when interpreting the sequence shown:

- Three items of software are involved:
  - the *browser wallet*: this runs in the client and contains details of all the payment cards the vendor currently holds;
  - the *vendor server*: this is the Web server and, in addition to responding to requests for product information, it manages the (electronic) purchase of items from the vendor's catalog;
  - the *acquirer gateway*: this is located in the vendor's bank computer and manages the payment phase of a purchase;
- For each new purchase, the vendor allocates a unique transaction identifier (TI) and this is included in all subsequent messages.
- During the order and purchase phases, the order information sent to the vendor contains only the card name and it is the purchase information that is sent to the vendor's bank that contains the actual card number.
- The payment request is carried out using existing inter-bank fund transfer procedures.

Further information about SET can be found in the bibliography for this chapter.

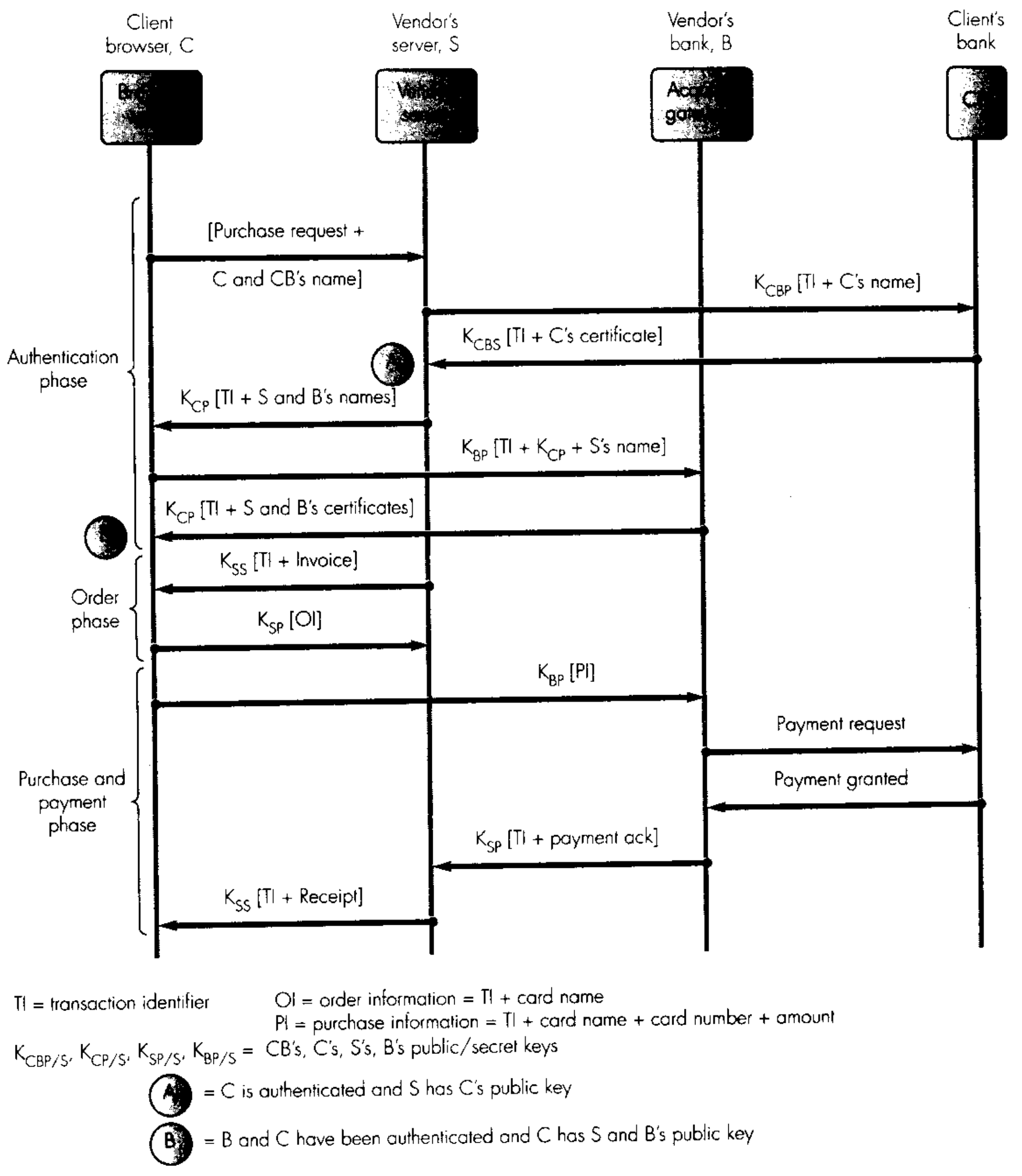

Client browser, C
Vendor's server, S
Vendor's bank, B
Client's bank
Authentication phase
[Purchase request + C and CB's name]
K_{CBP} [TI + C's name]
K_{CBS} [TI + C's certificate]
A
K_{CP} [TI + S and B's names]
K_{BP} [TI + K_{CP} + S's name]
K_{CP} [TI + S and B's certificates]
B
Order phase
K_{SS} [TI + Invoice]
K_{SP} [OI]
Purchase and payment phase
K_{BP} [PI]
Payment request
Payment granted
K_{SP} [TI + payment ack]
K_{SS} [TI + Receipt]
TI = transaction identifier
OI = order information = TI + card name
PI = purchase information = TI + card name + card number + amount
$K_{CBP/S}$, $K_{CP/S}$, $K_{SP/S}$, $K_{BP/S}$ = CB's, C's, S's, B's public/secret keys
A = C is authenticated and S has C's public key
B = B and C have been authenticated and C has S and B's public key

Figure 15.22 Outline of the operation of secure electronic transactions (SET).

15.7 Web operation

So far in this chapter we have described a range of topics relating to how a Web page is created and transferred over the Internet. In the last section we saw how transactions over the Internet are made secure. In this section we describe a number of issues relating to how the Web is organized and its key operational parts.

The first issue is how the presence of a server with a new set of pages/documents becomes known to users of the Web. There are a number of ways this can be done. For example, as we saw earlier in Section 15.2.1, the URL of the home page (together with a description of what the page is offering) can be posted to a related newsgroup within UseNet using the *news:* protocol/method. Alternatively, a more popular method is to use what is called a **search engine** together with a special browser called a **spider** (or **robot**).

The volume of information on the Web is already vast and is increasing continuously. In practice, however, only a small subset of this information is of interest to a particular user. The second issue, therefore, is how a user gains access to this subset of information without having to search the entire Web. This is done using an intermediary called a **portal**. We shall describe each of these issues separately.

15.7.1 Search engines

Before we can describe how a new Web page is made available over the Web we must first build up an understanding of how the search process is carried out by a search engine to find and retrieve a Web page. As we saw in Section 15.2, each Web page/document is accessed using the page's URL which, since it contains the unique domain name of the server computer on which the page/file is stored, is unique within the entire Web/Internet. Hence providing we know the URL of a page, a browser can readily obtain and display the page contents. To help with this, all browsers allow the user to keep a list of URLs in a table and provide facilities for the user to select, add, and delete entries. Nevertheless, in many instances, when searching for information on the Web, the user does not have a URL but rather he or she is interested in, say, any pages relating to a specific topic/subject. As a result, a directory is required. This is analagous to a telephone directory since, for each entry, a Web directory contains some information that describes the contents of the page plus its URL. A key issue is what this information should be and how it is used to find a specific URL.

A second issue relates to the size of the directory. As we have said, the number of Web pages is already vast and is increasing continuously. It would be totally infeasible to have a single large directory since the time required to carry out each search operation would be endless. This also applies to the telephone directory system of course and, to make searching the latter faster, the directory is fragmented into many separate sections. Typically, this is based on geographic location and, at a local level, the directory is divided into business and residential subscribers. When the telephone number of a customer is required, in addition to the customer's name, the location of the customer and whether it is a business or private residence is requested. In this way the search for the given name is restricted to a small subsection of the total directory.

The same approach is followed for the Web directory except the partitioning of the directory is not as straightforward. As we indicated earlier, with the Web the search information/index is based on a given topic/subject which is much less precise than a given name and location. For example, since each page has a title field, in principle, this could be used as the search index for the page. In practice, this cannot be done since in many instances a title is not given and, when one is given, it often bears no relation to the actual page contents. Instead, therefore, for each page, an additional block of information – normally in the form of a string of **keywords** – that describes the contents of the page is defined and it is these that are used as search indexes.

In practice, the choice of keywords to go with each URL varies widely and, as a result, there are a large number of different search engines in existence. In general, however, most allow a user to add the URL of the home page of a new set of pages to their current set of directories. Normally, this is done by first accessing the home page of the company that manages the search engine and, through this, a fill-in form is obtained. This is then filled in by the user and, when it is submitted, the page is added to the related directory. Alternatively, there are commercial organizations which, when a new URL is submitted, will add this to a number of search engines for you.

Spiders and robots

There are two different ways to obtain the search indexes for each page. One involves the person who submits the page providing a set of keywords in the same fill-in form that is submitted/posted to the owner of the search engine. These are then used to determine into which directory the URL should be inserted. In the second method, only the URL is submitted and it is the company which manages the search engine that obtains the keywords. It does this by using a special browser called a spider – also called a robot or simply **bot** – which, when given the submitted URL, accesses and then searches the contents of the page for specific keywords. It then uses these to enter the URL into the most appropriate directory(ies). The spider then follows all the links from the submitted home page and derives a set of keywords for each of these pages too. In addition, to make people aware that a new set of pages is now available, most search engine companies allow the URL of the home page (together with an associated set of keywords) to be posted to their **What's New** site. Normally, the page is then kept in the What's New directory for a set period of time.

Since in the second method it is the spider/robot that determines the set of keywords to be used, these may not be an optimum set as seen by the writer of the page. Hence when a spider is used, it is possible to direct the spider by including in the page header a list of the keywords that the writer feels should be used. The given list of keywords is called the **meta information** for the page and is included in the page header between the <HEAD> and </HEAD> tags. The list of keywords is given within the <META> tag using the following arguments:

- NAME = "keywords": to allow for other types of meta information, this informs the HTML interpreter that what follows is a list of keywords;
- LANG = "language": the language used for the keywords, for example en-US, es (Spanish), fr (French);
- CONTENT = "list of keywords": the list of keywords with a space between each.

An example format is:

```
<META NAME      = "keywords"
      LANG      = "en-US"
      CONTENT   = "vacations holidays walking beach activity hotels
                   scubadiving - - -">
```

15.7.2 Portals

As we indicated earlier, there are many different search engine companies/sites available which, given a set of keywords, will carry out a search through their directories and return details of a list of up to, say, 10 Web sites that give the best matches with a given set of keywords. Typically, the returned details for each site are in the form:

Match:	"80%"
Title:	"The name of the Web site"
URL:	"The URL of the home page"
Summary:	"A summary of what this site has to offer"
More:	"Click on this link to initiate a search for more pages like this one"

and are listed according to their match field.

To help you find the best search engine sites, most browsers have a button on the display which, when clicked, returns a list of a number of sites together with some information about what each has to offer. In addition, many Internet service providers (ISPs) provide a facility to help a user find the best search engine site(s) for a given set of keywords. This is referred to as the ISP's portal since it acts as a gateway to the vast collection of Web sites/pages that are now available.

Essentially, the portal has knowledge of a large collection of search engine sites and their directories. Given a set of keywords, the portal will select the site that it thinks has the best directory relating to the query. In addition, some portals will interact with the user through a form to obtain a more focussed set of requirements before returning the URL of what it thinks is the best site. Some ISPs also allow a user to create his or her own personalized portal. Then, by simply clicking on the portal page, the user is able to access his or her own preferred sources of information relating to, say, news, sport, weather, entertainment, and so on.

Summary

In this chapter we have described selected aspects of the operation of the World Wide Web. The list of topics is summarized in Figure 15.23. They can be grouped under the following headings:

- how a Web page is written in HTML and accessed over the Internet;
- how audio and video are incorporated into a Web page and accessed in real time using either a Web server or a streaming server. Also, how the user can control the way the audio and/or video is played out using the real-time streaming protocol (RTSP);
- an introduction to Java and JavaScript and the features each provides;
- security in e-commerce applications using the secure socket layer (SSL) and secure electronic transactions (SET) protocols;
- an introduction to the operation of the Web including the role of search engines and portals.

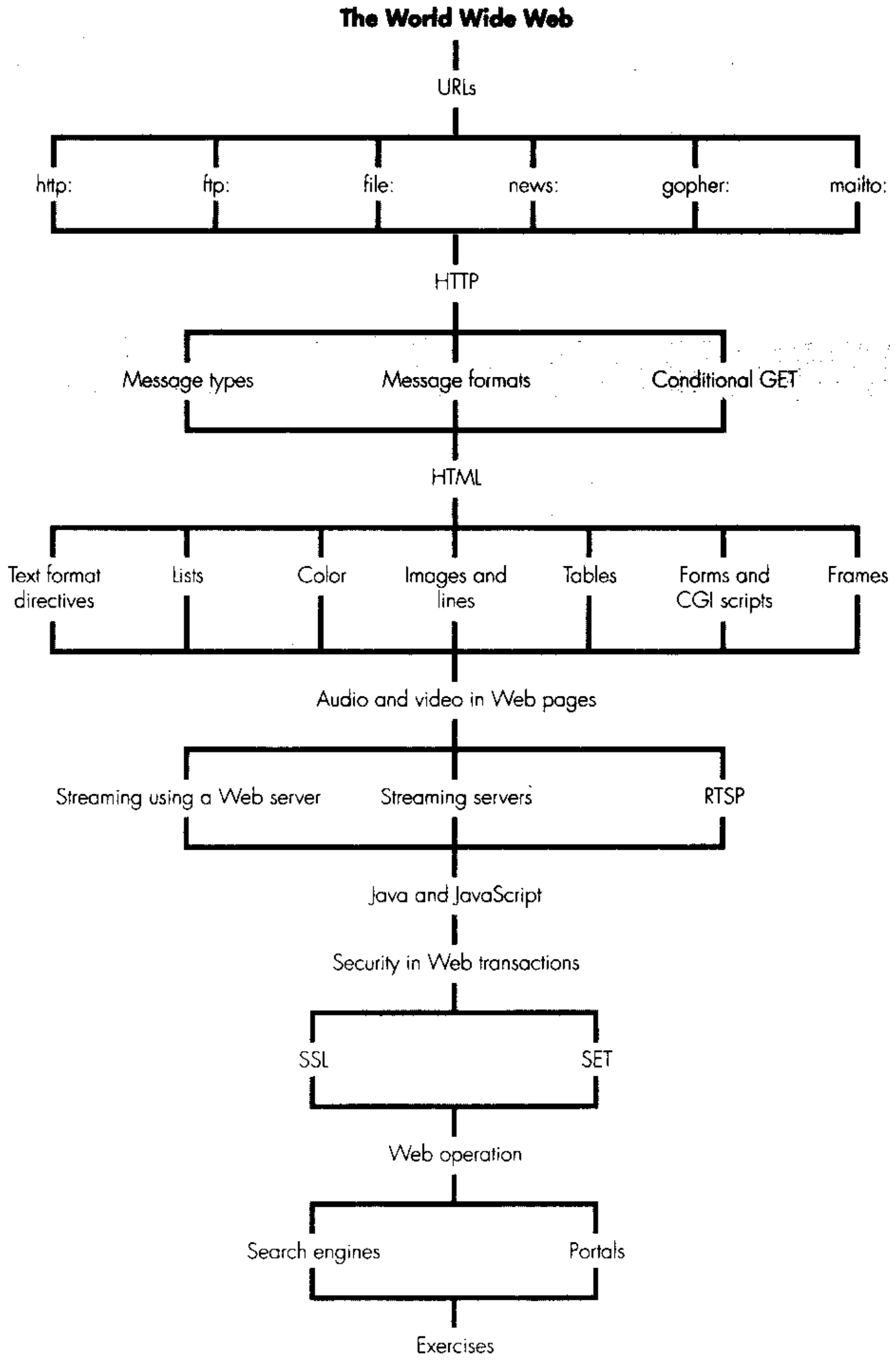


Figure 15.23 Summary of the topics discussed relating to the Web.

Exercises

Section 15.2

15.1 With the aid of a diagram, in a few sentences, explain each of the following terms relating to the Web: a browser, a server, HTTP, HTML, hyperlinks, URLs.

15.2 Assuming the URL of a Web page is:

http://www.UoW.edu/prospectus/index.html

identify the application protocol/method that is used to access the page, the domain name of the server, the pathname of the file, and the file name.

15.3 Assuming the URL

ftp://www.mpeg.org/mpeg-4/

determine the directory and file name of the page.

15.4 Assuming a user clicks on the URL

ftp://www.mpeg.org/mpeg-4

explain what type of information is returned.

15.5 Give an example of a URL that uses the *file:* method and explain one of its uses.

15.6 Outline the operation of UseNet including the role of newsgroups, a news reader, and NNTP. Hence explain how the home page of a new set of linked pages could be announced from within a Web page to a related newsgroup over UseNet.

15.7 Outline the steps that are followed by the browser to enable a user to send an email message from within a Web page.

15.8 Explain the meaning of the term "URI" and how it differs from a URL.

15.9 With the aid of a diagram, show the protocol stack that is used in a Web client and Web server to obtain a page/block of data using the HTTP application protocol/method. Include in your diagram a DNS name server and explain its role in relation to obtaining the page.

15.10 Describe how it is possible to obtain satisfactory performance with a simple request response application protocol for transferring a page over the Web.

15.11 List the advantages and disadvantages of using the following types of TCP connections for a Web session:
(i) nonpersistent connections,
(ii) multiple concurrent nonpersistent connections,
(iii) persistent connections.

15.12 In relation to HTTP, state the difference between a simple request message and a full request message. How does HTTP discriminate between the two message types?

15.13 Using the example HTTP request/response messages shown in Figure 15.3:
(i) state the implications of the presence of the *HTTP/1.1* and *Connect:close* fields in the request message
(ii) if the response message related to, say, a JPEG image rather than an HTML page, give a typical set of content-related fields.

15.14 Explain the role of a cache server and how its use can speed up the time to obtain a Web page. Clearly indicate where the savings arise and how they can be reduced further by using a hierarchy of cache servers.

15.15 State how a conditional GET request message differs from a GET request. Hence, with the aid of a diagram, illustrate the message sequence that is followed when a browser obtains the contents of a named file from a named (origin) server via a proxy/cache server. Assume the contents of the file have been modified since the date given in the conditional GET request message.

Section 15.3

15.16 With the aid of an example, explain the terms "absolute URL" and "relative URL" including the relationship between the two.

15.17 In relation to the HTML script shown in Figure 15.5, assume the contents of the page accessed through the link *The University* are as shown in the right frame of 15.13(a). Assuming frames are not being used, write the HTML script for the portion of the page that is displayed.

15.18 In relation to the HTML script shown in Figure 15.5, show the changes to the script if the complete prospectus was on a single page rather than a linked set of pages. State the advantages and disadvantages of doing this.

15.19 Assuming the index for this book is to be on a single Web page, write the fragment of HTML script for the first five entries in the list of contents for this chapter using the <OL> tag.

15.20 In order to enhance your Web page, you decide to introduce color into it. Show how you could direct the HTML interpreter in a browser to make the background color yellow, the text in the page orange, each link red, and a visited link blue.

15.21 In relation to the two displayed images shown in Figure 15.7, produce a segment of HTML script:
(i) to display the UoW crest/logo in the center of the page with the first-level heading starting below it – part (a),
(ii) to start the text at the top edge of the text – part (b)
(iii) to make the image shown in part (b) a hyperlink.

15.22 Give the additions to the HTML script shown in Figure 15.5 to produce a bold line that divides the heading from the remaining text.

15.23 Assume the table shown in Figure 15.9(b) is to be changed so that there are four columns CO, CLS, CBR, and VBR with a + character in those column positions where the related feature is true. For example, for the LAN row, the CLS and VBR columns would each have a + character and the other two columns would be left blank. Produce the HTML script for the table.

15.24 Give the changes to the HTML script shown in Figure 15.9(c) to align the contents of the cells in the NETWORK column to the left edge of the cell.

15.25 Give the outline of an HTML script to produce a table that comprises 3 columns and 3 rows with the second and third cells of the first column combined and the second and third cells of the second row combined. Assume the 3 cells in the first row are for headings and the contents of all cells are to be left blank.

15.26 (i) State the purpose of a fill-in form and an associated CGI script.
(ii) Use an example FORM declaration to explain the use of the ACTION and METHOD attributes.
(iii) List some of the alternative ways input can be obtained from a user.

15.27 Use the <INPUT> tag with appropriate attributes to create the following form. Include in your HTML script an example URL for the ACTION attribute.

ICE CREAM ORDER FORM
Name:
Address:

Phone number:
Please check flavors:
Vanilla ☐ Chocolate ☐ Strawberry ☐
Please indicate how many boxes you require:
One ☐ Two ☐ Three ☐
Payment card Type: MC ☐ Visa ☐
Number: Expiry date:
Submit order Reset order

15.28 List a typical set of responses for the various fields in the form shown in Exercise 15.27. Explain how these are sent to a CGI script/program in the server given in your example URL.

15.29 Show how the choice of flavors in your HTML script for Exercise 15.27 could be presented using a pull-down menu.

15.30 Assume you want to create an album for your digitized photographs on your own computer which can be viewed through the browser. Write a segment of an HTML script for a page template that divides the display window into four quarters using frames. Include in your script the URL of the image/photograph you want to display in each frame.

15.31 Explain the meaning and use of an in-line frame. Write a segment of an HTML script that shows how a second frame can be used to display the contents of a page that is referenced in the current page.

Section 15.4

15.32 When downloading a Web page comprising audio and/or video, by means of examples, explain why streaming and a playout buffer are used. Hence list and explain the functions performed by a media player.

15.33 With the aid of a diagram, explain how a file containing a short video clip – comprising audio and video – is streamed from a Web server and played out by a browser that has both an audio and a video media player. Include in your diagram the protocols that are used to transfer the media.

15.34 With the aid of a diagram, explain how the playout process used in the last exercise can be improved by using a meta file. Include in your diagram the protocols that are used to transfer the meta file and the media streams.

Identify the limitations associated with this approach.

15.35 With the aid of a diagram, explain how a movie/video is accessed from a streaming server and played out by a browser. Include in your diagram the protocols that are used at both the server side and the client side including, in the case of the various application protocols, their port numbers and use. Include in your explanation the sequence of steps that occur when the user first selects a movie/video.

15.36 List a typical set of control buttons associated with the playout of a movie/video on the screen of a TV/PC. Hence use a diagram to illustrate the application protocol and control/data parts of the media player that is involved when the user carries out the following sequence:
(i) initiates the showing of a movie/video
(ii) activates the play button
(iii) activates the pause button
(iv) activates the quit button.

Section 15.5

15.37 Both Java and JavaScript can be used to add action and interactivity to a Web page. Explain briefly how this is done in each case.

15.38 In relation to the Java programming language, explain the meaning of the following terms:
(i) an applet,
(ii) bytecode,
(iii) bytecode interpreter.

15.39 Give an example segment of HTML script that uses the <APPLET> tag. Explain the actions that are followed by the HTML parser and bytecode interpreter when the tag is encountered in the script. Include in your description an example URL for the page and the name of the file associated with the <APPLET> tag.

15.40 In HTML 4.0 and later versions, the <OBJECT> tag is used in preference to the <APPLET> tag. Explain why this change has come about. Give an example declaration of a Java applet using the <OBJECT> tag and an associated set of attributes. Clearly identify the role performed by each attribute.

15.41 Explain the meaning of the following terms relating to the Java programming language:
(i) an object,
(ii) a method,
(iii) encapsulation,

(iv) object class,
(v) package.

Give two examples of a Java library package and explain their function.

15.42 Like Java, JavaScript is object-oriented. Show how the *document* object class and the associated *.write*, *.URL* and *.title* methods can be used in a JavaScript that is embedded in an HTML page.

15.43 By means of examples, explain the meaning of the term "event handler" in the context of the JavaScript programming language. Hence explain how an event handler can be used in conjunction with a form to perform a specific action when the event occurs. Use as an example a form that takes as input the name and password entered by a user and performs checks on these.

Section 15.6

15.44 Give three examples of security threats that illustrate the need for additional security measures in e-commerce and e-banking applications.

15.45 In relation to the secure socket layer (SSL) scheme, with the aid of a diagram, explain the function of the various protocols that are used to carry out a secure transaction. Include in your description how the HTML interpreter differentiates between a secure and an insecure transaction.

15.46 To carry out a secure transaction using the SSL scheme, the following five steps are required:
(i) authentication using a certification authority
(ii) cryptographic algorithm negotiation
(iii) session key exchange
(iv) transaction initiation
(v) transaction information transfer.

With the aid of a diagram and selected keys, illustrate how each of the above steps is carried out.

15.47 Identify a potential security loophole with the SSL scheme and state how the secure electronic transactions (SET) scheme overcomes this.

15.48 Identify the four main players that are involved in the SET scheme and give a brief description of the role of the software associated with each of them.

15.49 Assuming public key cryptography throughout, use a diagram to show how:
(i) the client authenticates the vendor and vice versa
(ii) the client sends an order to the vendor
(iii) the client carries out the purchase and payment operations.

Section 15.7

15.50 Outline the role of a search engine. Identify the two main issues that influence its design and explain why keywords are used.

15.51 Explain the role of a spider/robot in relation to a search engine and how this can be influenced by the writer of a page by providing meta information. By means of an example, show how the latter is included in a Web page using the <META> tag and the NAME, LANG, and CONTENT attributes.

15.52 By means of an example, explain the structure of the information that is returned from a search operation. Also explain the role of a portal in carrying out a search.