UNIT – I

Overview

This section provides information about web technologies that relate to the interface between web servers and their clients. This information includes markup languages, programming interfaces and languages, and standards for document identification and display. The term "**Web 2.0**" (2004–present) is commonly associated with web applications that facilitate interactive information sharing, interoperability, user-centered design and collaboration on the World Wide Web. Examples of Web 2.0 include web-based communities, hosted services, web applications, social-networking sites, video-sharing sites, wikis, blogs, mashups, and folksonomies. A Web 2.0 site allows its users to interact with other users or to change website content, in contrast to non-interactive websites where users are limited to the passive viewing of information that is provided to them.

Objectives:

- Explain about the principles of internet
- Explain a basic web concepts
- Explain how does client sever model work
- what is marup language and how it is embedded in web

1. INTRODUCTION

1.1. INTERNET PRINCIPLES

This page reviews the central concepts of software on the internet. It briefly explains:

- **TCP/IP**
- **UDP**
- **IP Addresses**
- **domain names**
- **the domain name system**
- **ports**
- **sockets**
- **URL's**

1.1. 1. TCP/IP

The Internet is the network that connects computers all over the world. It works according to a set of agreed protocols. **TCP (Transmission Control Protocol)** and **IP(Internet Protocol)** are the most commonly-used protocols for using the Internet. (But there are others at lower levels.) The combination is simply known as **TCP/IP.**

The Internet is a **packet switching** system. Any message is broken into packets that are transmitted independently across the interment (sometime by different routes). These packets are called **datagrams.** The route chosen for each datagram depends on the traffic at any point in time. Each datagram has a header of between 20 and 60 bytes, followed by the payload of up to 65,515 bytes of data. The header consists of, amongst other data:

1. the version number of the protocol in use
2. IP address of sender
3. IP address of destination

TCP breaks down a message into packets. At the destination, it re-assembles packets into messages. It attaches a checksum to each packet. If the checksum doesn't match the computed checksum at the destination, the packet is re-transmitted. Thus TCP ensures reliable transmission of information. In summary, TCP:

1. provides re-transmission of lost data
2. delivery of data in the correct order

1.1.2. UDP

Most applications use TCP. However, an example of a situation in which it is desirable to use a lower-level protocol is the case of audio **streaming.** If you want to download a sound file, it can take some time, even though it may be compressed. You have to wait (maybe some time) for the complete file to download before it can be played. An alternative is to listen to the sound as it is being downloaded - streaming. One of the most popular technologies is called RealAudio.

RealAudi does not use TCP because of its overhead. The sound file is sent in IP packets using the **UDP (User Datagram Protocol)** instead of TCP. UDP is an unreliable protocol:

- it doesn't guarantee that a packet will arrive

- it doesn't guarantee that packets are in the right order

UDP doesn't re-send a packet if it is missing or there is some other error, and it doesn't assemble packets into the correct order. But it is faster than TCP. In this application, losing a few bits of data is better than waiting for the re-transmission of some missing data. The application's major mission is to keep playing the sound without interruption. (In contrast, the main goal of a file transfer program is to transmit the data accurately.)

The same mechanism is used with video streaming.

UDP is a protocol at the same level as TCP, above the level of IP.

1.1.3. Domain name

The **domain name** is the user-friendly equivalent of an IP address. Used because the numbers in an IP address are hard to remember and use. Also known as a host name. Example:

shu.ac.uk

Such a name starts with the most local part of the name and is followed by the most general. The whole name space is a tree, whose root has no name. The first level in the tree has com, org, edu, UK, etc.

The parts of a domain name don't correspond to the parts of an IP address. Indeed

domain names don't always have 4 parts - they can have 2, 5 or whatever.

All applications that use an address should work whether an IP address or a domain name is used. In fact, a domain name is converted to an IP address before it is used.

Exercise: Compare and contrast IP addresses with domain names.

1.1.4. Domain Name System

A program, say a Web browser, that has a domain address usually needs to convert it into an IP address before making contact with the server. The domain name system (DNS) provides a mapping between IP addresses and domain names. All this information cannot be all in one place and so it is a distributed database.

1.1.5. Clients, Servers and Peers

A network application usually involves a client and a server. Each is a process (an independently running program) running on a (different) computer.

A server runs on a host and provides some particular service, e.g. e-mail, access to local Web pages. Thus a Web server is a server. A commonly-used web server program is called Apache.

A client runs on a host but needs to connect with a sever on another host to accomplish its task. Usually, different clients are used for different tasks, e.g. Web browsing and e-mail. Thus a Web browser is a client.

Some programs are not structured as clients and servers. For example a game, played across the internet by two or more players is a peer to peer relationship. Other examples: chat, internet phone, shared whiteboard.

1.1.6. Port Numbers

To identify a host machine, an IP address or a domain name is needed. To identify a particular server on a host, a port number is used. A port is like a logical connection to a machine. Port numbers can take values from 1 to 65,535. It has no correspondence with the physical connections, of which there might be just one. Each type of service has, by convention, a standard port number. Thus 80 usually means Web serving and 21 means file transfer. If the default port number is used, it can be omitted in the URL (see below). For each port supplying a service there is a server program waiting for any requests. Thus a web server program listens on port 80 for any incoming requests. All these server programs run together in parallel on the host machine.

When a packet of information is received by a host, the port number is examined and the packet sent to the program responsible for that port. Thus the different types of request are distinguished and dispatched to the relevant program.

The following table lists the common services, together with their normal port numbers. These conventional port numbers are sometimes not used for a variety of reasons. One example is when a host provides (say) multiple web servers, so only one can be on port 80. Another reason is where the server program has not been assigned the privilege to use port 80.

1.1.7. Sockets

A socket is the software mechanism for one program to connect to another. A pair of programs opens a socket connection between themselves. This then acts like a

telephone connection - they can converse in both directions for as long as the connection is open. (In fact, data can flow in both directions at the same time.) More than one socket can use any particular port. The network software ensures that data is routed to or from the correct socket.

When a server (on a particular port number) gets an initial request, it often spawns a separate thread to deal with the client. This is because different clients may well run different speeds. Having one thread per client means that the different speeds can be accommodated. The new thread creates a (software) socket to use as the connection to the client. Thus one port may be associated with many sockets.

1.1.8. Streams

Accessing information across the Internet is accomplished using streams. A stream is a serial collection of data, such as can be sent to a printer, a display or a serial file. Similarly a stream is like data input from a keyboard or input from a serial file. Thus reading or writing to another program across a network is just like reading or writing to a serial file.

1.1.9. URL

A URL (Uniform Resource Locator) is:

- a unique identifier for any resource on the Internet
- typed into a Web browser
- used as a hyperlink within a HTML document
- quoted as a reference to a source

A URL has the structure:

protocol: //hostname[:port]/[pathname]/filename#section

The host name is the name of the server that provides the service. This can either be a domain name or an IP address.

The port number is only needed when the server does not use the default port number. For example, 80 is the default port number for HTTP.

1.2. BASIC WEB CONCEPTS

This section describes the essential concepts for working with PowerDynamo (Storing, locating, and transmitting information on the Web)

How information is located: the URL

To move from one page of a document to another page, or to another document on the same or another Web site, the user clicks a hyperlink (usually just called a link) in the document shown in their Web client. Documents and locations within documents are identified by an address, defined as a Uniform Resource Locator, or **URL**. The following URL illustrates the general form:

http://www.sybase.com/productsl

or

http://www.sybase.com/inc/corpinfo/mkcreate.html

URLs contain information about which server the document is on, and may also specify a particular document available to that server, and even a position within the document. In addition, a URL may carry other information from a Web client to a Web server, including the values entered into fields in an HTML form.

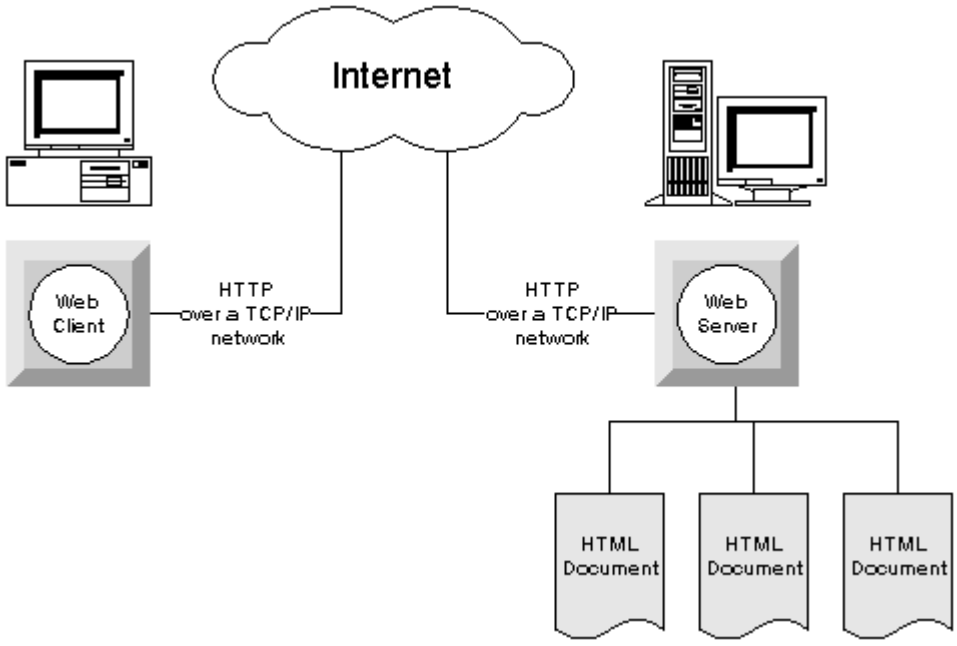
For more information about URLs and addresses on the Web, see the material on the World Wide Web Consortium pages, at the following address:

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

When a user clicks a link on a document on their Web client, the URL is sent to the server of the indicated Web site. The Web server locates the document, and sends the HTML to the Web client across the network.

The below figure illustrates, information is stored at **Web sites**. Access to the information is managed by a **Web server** for the site. Users access the information using **Web clients**, which are also called **browsers**.

Figure 2-1: Accessing information on the Web



Information on the Web is stored in documents, using a language called **HTML** (HyperText Markup Language). Web clients must interpret HTML to be able to display the documents to a user. The protocol that governs the exchange of information between the Web server and Web client is named **HTTP** (HyperText Transfer Protocol).

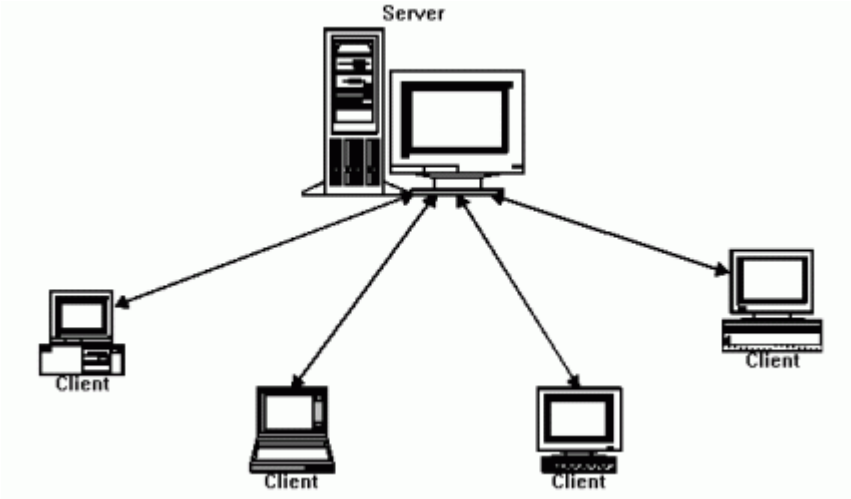
External data in HTML documents

HTML documents can include graphics or other types of data by referencing an external file (for example, a GIF or JPEG file for a graphic). Not all these external formats are supported by all Web clients. When the document contains such data, the Web client can send a request to the Web server to provide the relevant graphic. If the Web client does not support the format, it does not request the information from the server.

1.3. CLIENT / SERVER MODEL

1.3.1. Client server

Client/server describes the relationship between two computer programs in which one program, the client, makes a service request from another program, the server, which fulfils the request. Although programs within a single computer can use the client/server idea, it is a more important idea in a network. In a network, the client/server model provides a convenient way to interconnect programs that are distributed efficiently across different locations. Computer transactions using the client/server model are very common. For example, to check your bank account from your computer, a client program in your computer forwards your request to a server program at the bank. That program might in turn forward the request to its own client program that sends a request to a database server at another bank computer to retrieve your account balance. The balance is returned back to the bank data client, which in turn serves it back to the client in your personal computer, which displays the information for you.



Client server model diagram

Source: http://www.javaworld.com/javaworld/jw-10-2001/jw-1019-jxta.html -

The communications method in computing includes Local procedure calls and Remote procedure calls.

1.3.2. Remote Procedure Call:

This is a protocol that one program can use to request the services from other located in other machine in a network without having to understand the network details. Usually when a program using RPC are compiled into an executable program, a stub is included which acts as the representative of remote procedure code. When the program is run, and the procedure issue the stub receives a request and forwards it to the client runtime in the local computer by the daemons. The client runtime program knows the address of the remote computer and server application. It then sends the request across the network .The server also have a runtime program and stub that interface with remote procedure. The result is returned the same way.

1.3.3. Local Procedure Call

A local procedure call (LPC) is an interprocess communication facility for high-speed message passing

In Windows NT, client-subsystem communication happens in a fashion similar to that in the MACH operating system. Each subsystem contains a client-side DLL that links with the client executable. The DLL contains stub functions for the subsystem's API. Whenever a client process–an application using the subsystem interface–makes an API call, the corresponding stub function in the DLL passes on the call to the subsystem process. The subsystem process, after the necessary processing, returns the results to the client DLL. The stub function in the DLL waits for the subsystem to return the results and, in turn, passes the results to the caller. The client process simply resembles calling a normal procedure in its own code. In the case of RPC, the client actually calls a procedure sitting in some remote server over the network–hence the name remote

procedure call. In Windows NT, the server runs on the same machine; hence the mechanism is called as a local procedure call.

LPC is designed to allow three methods of exchanging messages:

- Message that is shorter than 256 bytes can be sent by calling LPC with a buffer containing the message. That is a small message. This message is then copied from the address space of the sending process into system address space, and then to the address space of the receiving process.

- If a client and a server want to exchange more than 256 bytes of data, they can choose to use a shared section to which both are mapped. The sender places message data in the shared section and then sends a small message to the receiver with pointers to where the data is to be found in the shared section.

- When a server wants to read or write larger amount of data than will fit in a shared section, data can be directly read from or written to a client's address space. The LPC component supplies two functions that a server can use to accomplish this. A message sent by the first method is used to synchronize the message passing.

There are three types of LPC. The first type sends small messages up to 304 bytes. The second type sends larger messages. The third type of LPC is called as Quick LPC and used by the Win32 subsystem in Windows NT 3.51.

The first two types of LPC use port objects for communication. Ports resemble the sockets or named pipes in Unix. A port is a bi-directional communication channel between two processes. However, unlike sockets, the data passed through ports is not

streamed. The ports preserve the message boundaries. Simply put, you can send and receive messages using ports. The subsystems create ports with well-known names. The client processes that need to invoke services from the subsystems open the corresponding port using the well-known name. After opening the port, the client can communicate, with the server, over the port.

1.4 HTML & SCRIPTING LANGUAGES

HTML, an initializes for **Hypertext Mark-up Language**, is the predominant markup language for web pages. It provides a means to describe the structure of text-based information in a document—by denoting certain text as links, headings, paragraphs, lists, etc.—and to supplement that text with interactive forms, embedded images, and other objects. HTML is written in the form of "tags" consisting minimally of "elements" surrounded by angle brackets. HTML can also describe, to some degree, the appearance and semantics of a document, and can include embedded scripting language code (such as JavaScript) that can affect the behavior of Web browsers and other HTML processors. History of HTML

Origins

In 1980, physicist Tim Berners-Lee, who was an independent contractor at CERN, proposed and prototyped ENQUIRE, a system for CERN researchers to use and share documents. In 1989, Berners-Lee and CERN data systems engineer Robert Cailliau each submitted separate proposals for an Internet-based hypertext system providing similar functionality. The following year, they collaborated on a joint proposal, the WorldWideWeb (W3) project, which was accepted by CERN. In his personal notes from 1990 he lists, "some of the many areas in which hypertext is used", and puts an encyclopaedia first.

First specifications

The first publicly available description of HTML was a document called HTML Tags, first mentioned on the Internet by Berners-Lee in late 1991. It describes 22 elements comprising the initial, relatively simple design of HTML. Thirteen of these elements still exist in HTML 4. HTML is a text and image formatting language used by web browsers to dynamically format web pages. The semantics of many of its tags can be traced to early text formatting languages such as that used by the RUNOFF command developed in the early 1960s for the CTSS (Compatible Time-Sharing System) operating system, and its formatting commands were derived from the commands used by typesetters to manually format documents.

Berners-Lee considered HTML to be, at the time, an application of SGML, but it was not formally defined as such until the mid-1993 publication, by the IETF, of the first proposal for an HTML specification: Berners-Lee and Dan Connolly's "Hypertext Markup Language (HTML)" Internet-Draft, which included an SGML Document Type Definition to define the grammar. The draft expired after six months, but was notable for its acknowledgment of the NCSA Mosaic browser's custom tag for embedding in-line images, reflecting the IETF's philosophy of basing standards on successful prototypes. Similarly, Dave Raggett's competing Internet-Draft, "HTML+ (Hypertext Markup Format)", from late 1993, suggested standardizing already-implemented features like

tables and fill-out forms.

After the HTML and HTML+ drafts expired in early 1994, the IETF created an HTML Working Group, which in 1995 completed "HTML 2.0", the first HTML specification intended to be treated as a standard against which future implementations should be based. Published as Request for Comments 1866, HTML 2.0 included ideas from the HTML and HTML+ drafts. There was no "HTML 1.0"; the 2.0 designation was intended to distinguish the new edition from previous drafts.

Further development under the auspices of the IETF was stalled by competing interests. Since 1996, the HTML specifications have been maintained, with input from commercial software vendors, by the World Wide Web Consortium (W3C). However, in 2000, HTML also became an international standard (ISO/IEC 15445:2000). The last HTML specification published by the W3C is the HTML 4.01 Recommendation, published in late 1999. Its issues and errors were last acknowledged by errata published in 2001.

HTML version timeline

November 1995

HTML 2.0 was published as IETF RFC 1866. Supplemental RFCs added capabilities:

- November 1995: RFC 1867 (form-based file upload)
- May 1996: RFC 1942 (tables)
- August 1996: RFC 1980 (client-side image maps)
- January 1997: RFC 2070 (internationalization)

In June 2000, all of these were declared obsolete/historic by RFC 2854.

January 1997

HTML 3.2 was published as a W3C Recommendation. It was the first version developed and standardized exclusively by the W3C, as the IETF had closed its HTML Working Group in September 1997.
HTML 3.2 dropped math formulas entirely, reconciled overlap among various proprietary extensions, and adopted most of Netscape's visual markup tags. Netscape's blink element and Microsoft's marquee element were omitted due to a mutual agreement between the two companies. A markup for mathematical formulas similar to that in HTML wasn't standardized until 14 months later in MathML.

December 1997

HTML 4.0 was published as a W3C Recommendation. It offers three "flavors":

- Strict, in which deprecated elements are forbidden,
- Transitional, in which deprecated elements are allowed,
- Frameset, in which mostly only frame related elements are allowed;

Initially code-named "Cougar", HTML 4.0 adopted many browser-specific element types and attributes, but at the same time sought to phase out Netscape's visual markup features by marking them as deprecated in favor of style sheets.

April 1998

HTML 4.0 was reissued with minor edits without incrementing the version number.

December 1999

HTML 4.01 was published as a W3C Recommendation. It offers the same three flavors as HTML 4.0, and its last errata were published May 12, 2001.

May 2000

ISO/IEC 15445:2000 ("ISO HTML", based on HTML 4.01 Strict) was published as an ISO/IEC international standard.

As of mid-2008, HTML 4.01 and ISO/IEC 15445:2000 are the most recent versions of HTML. Development of the parallel, XML-based language XHTML occupied the W3C's HTML Working Group through the early and mid-2000s.

Drafts

October 1991

HTML Tags, an informal CERN document listing twelve HTML tags, was first mentioned in public. November 1992.

July 1993

Hypertext Markup Language was published by the IETF as an Internet-Draft (a rough proposal for a standard). It expired in January 1994.

November 1993

HTML+ was published by the IETF as an Internet-Draft and was a competing proposal to the Hypertext Markup Language draft. It expired in May 1994.

April 1995 (authored March 1995)

HTML 3.0 was proposed as a standard to the IETF, but the proposal expired five months later without further action. It included many of the capabilities that were in Raggett's HTML+ proposal, such as support for tables, text flow around figures, and the display of complex mathematical formulas.

A demonstration appeared in W3C's own Arena browser. HTML 3.0 did not succeed for several reasons. The pace of browser development, as well as the number of interested parties, had outstripped the resources of the IETF. Netscape continued to introduce HTML elements that specified the visual appearance of documents, contrary to the goals of the newly-formed W3C, which sought to limit HTML to describing logical structure. Microsoft, a newcomer at the time, played to all sides by creating its own tags, implementing Netscape's elements for compatibility, and supporting W3C features such as Cascading Style Sheets.

January 2008
HTML5 was published as a Working Draft by the W3C.

Although its syntax closely resembles that of SGML, HTML 5 has abandoned any attempt to be an SGML application, and has explicitly defined its own "html" serialization, in addition to an alternative XML

XHTML versions

XHTML is a separate language that began as a reformulation of HTML 4.01 using XML 1.0. It continues to be developed:

- XHTML 1.0, published January 26, 2000 as a W3C Recommendation, later revised and republished August 1, 2002. It offers the same three flavors as HTML 4.0 and 4.01, reformulated in XML, with minor restrictions.
- XHTML 1.1, published May 31, 2001 as a W3C Recommendation. It is based on XHTML 1.0 Strict, but includes minor changes, can be customized, and is reformulated using modules from Modularization of XHTML, which was published April 10, 2001 as a W3C Recommendation.
- XHTML 2.0, is still a W3C Working Draft. XHTML 2.0 is incompatible with XHTML 1.x and, therefore, would be more accurate to characterize as an XHTML-inspired new language than an update to XHTML 1.x.
- XHTML5, which is an update to XHTML 1.x, is being defined alongside HTML5 in the HTML5 draft.

1.4.1. HTML Elements

Remember the HTML example from the previous page:

```
<html>
<head>
<title>Title of page</title>
</head>
<body>
This is my first homepage. <b>This text is bold</b>
</body>
</html>
```

This is an HTML element:

```
<b>This text is bold</b>
```

The HTML element starts with a **start tag**:
The **content** of the HTML element is: This text is bold
The HTML element ends with an **end tag**:

The purpose of the tag is to define an HTML element that should be displayed as bold.

This is also an HTML element:

```
<body>
This is my first homepage. <b>This text is bold</b>
</body>
```

This HTML element starts with the start tag <body>, and ends with the end tag </body>.

The purpose of the <body> tag is to define the HTML element that contains the body of the HTML document.

1.4.2. Headings

Headings are defined with the <h1> to <h6> tags. <h1> defines the largest heading. <h6> defines the smallest heading.

```
<h1>This is a heading</h1>
<h2>This is a heading</h2>
<h3>This is a heading</h3>
<h4>This is a heading</h4>
<h5>This is a heading</h5>
<h6>This is a heading</h6>
```

HTML automatically adds an extra blank line before and after a heading.

1.4.3. Paragraphs

Paragraphs are defined with the tag.

```
<p>This is a paragraph</p>
<p>This is another paragraph</p>
```

HTML automatically adds an extra blank line before and after a paragraph.

1.4.4. Don’t Forget the Closing Tag

You might have noticed that paragraphs can be written without end tags :

```
<p>This is a paragraph
<p>This is another paragraph
```

The example above will work in most browsers, but don't rely on it. Future version of HTML will not allow you to skip ANY end tags.

Closing all HTML elements with an end tag is a future-proof way of writing HTML. It also makes the code easier to understand (read and browse) when you mark both where an element starts and where it ends.

1.4.5. Line Breaks

The
 tag is used when you want to break a line, but don't want to start a new paragraph. The
 tag forces a line break wherever you place it.

```
<p>This <br> is a para<br>graph with line breaks</p>
```

The
 tag is an empty tag. It has no end tag like </br>, since a closing tag doesn't make any sense.

**
 or
**

More and more often you will see the
 tag written like this:

Because the
 tag has no end tag (or closing tag), it breaks one of the rules for future HTML (the XML based XHTML), namely that all elements must be closed.

Writing it like
 is a future proof way of closing (or ending) the tag inside the opening tag, accepted by both HTML and XML.

1.4.6. Comments in HTML

The comment tag is used to insert a comment in the HTML source code. A comment will be ignored by the browser. You can use comments to explain your code, which can help you when you edit the source code at a later date.

```
<!-- This is a comment -->
```

Note that you need an exclamation point after the opening bracket, but not before the closing bracket.

1.4.7. HTML Tag Attributes

HTML tags can have attributes. Attributes provide additional information to an HTML element.

Attributes always come in name/value pairs like this: name="value".

Attributes are always specified in the start tag of an HTML element.

Attributes Example 1:

<h1> defines the start of a heading.

<h1 align="center"> has additional information about the alignment.

Attributes Example 2:

<body> defines the body of an HTML document.

<body bgcolor="yellow"> has additional information about the background color.

Attributes Example 3:

<table> defines an HTML table. (You will learn more about HTML tables later)

<table border="1"> has additional information about the border around the table.

Use Lowercase Attributes

Attributes and attribute values are case-insensitive. However, the World Wide Web Consortium (W3C) recommends lowercase attributes/attribute values in their HTML 4 recommendation, and XHTML demands lowercase attributes/attribute values.

1.4.8. Text Formatting Tags

Tag	Description
<b>	Defines bold text
<big>	Defines big text
	Defines emphasized text
<i>	Defines italic text
<small>	Defines small text
	Defines strong text
<sub>	Defines subscripted text
<sup>	Defines superscripted text
<ins>	Defines inserted text

<del>	Defines deleted text
<s>	Deprecated. Use <del> instead
<strike>	Deprecated. Use <del> instead
<u>	Deprecated. Use styles instead

"Computer Output" Tags

Tag	Description
<code>	Defines computer code text
<kbd>	Defines keyboard text
<samp>	Defines sample computer code
<tt>	Defines teletype text
<var>	Defines a variable
<pre>	Defines preformatted text
<listing>	Deprecated. Use <pre> instead
<plaintext>	Deprecated. Use <pre> instead
<xmp>	Deprecated. Use <pre> instead

Citations, Quotations, and Definition Tags

Tag	Description
<abbr>	Defines an abbreviation
<acronym>	Defines an acronym
<address>	Defines an address element
<bdo>	Defines the text direction
<blockquote>	Defines a long quotation
<q>	Defines a short quotation
<cite>	Defines a citation
<dfn>	Defines a definition term

1.4.9. Character Entities

Some characters have a special meaning in HTML, like the less than sign (<) that defines the start of an HTML tag. If we want the browser to actually display these characters we must insert character entities in the HTML source.

A character entity has three parts: an ampersand (&), an entity name or # and an entity number, and finally a semicolon (;).

To display a less than sign in an HTML document we must write: **<** or **<**

The advantage of using a name instead of a number is that a name is easier to remember. The disadvantage is that not all browsers support the newest entity names, while the support for entity numbers is very good in almost all browsers.

Note that the entities are case sensitive.

This example lets you experiment with character entities: Character Entities

---

1.4.10. Non-breaking Space

The most common character entity in HTML is the non-breaking space.

Normally HTML will truncate spaces in your text. If you write 10 spaces in your text HTML will remove 9 of them. To add spaces to your text, use the   character entity.

The Most Common Character Entities:

Result	Description	Entity Name	Entity Number
	non-breaking space		

<	less than	<	<
>	greater than	>	>
&	Ampersand	&	&
"	quotation mark	"	"
'	apostrophe	' (does not work in IE)	'

Some Other Commonly Used Character Entities:

Result	Description	Entity Name	Entity Number
¢	Cent	¢	¢
£	Pound	£	£
¥	Yen	¥	¥
€	Euro	€	€
§	Section	§	§
©	Copyright	©	©
®	registered trademark	®	®
×	multiplication	×	×
÷	Division	÷	÷

1.4.11. Anchor Tag and the Href Attribute

HTML uses the <a> (anchor) tag to create a link to another document.

An anchor can point to any resource on the Web: an HTML page, an image, a sound file, a movie, etc.

The syntax of creating an anchor:

```
<a href="url">Text to be displayed</a>
```

The <a> tag is used to create an anchor to link from, the href attribute is used to address the document to link to, and the words between the open and close of the anchor tag will be displayed as a hyperlink.

This anchor defines a link to W3Schools:

```
<a href="http://www.w3schools.com/">Visit W3Schools!</a>
```

The line above will look like this in a browser:

Visit W3Schools!

1.4.12. The Target Attribute

With the target attribute, you can define **where** the linked document will be opened.

The line below will open the document in a new browser window:

```
<a href="http://www.w3schools.com/"
Target="_blank">Visit W3Schools!</a>
```

The Anchor Tag and the Name Attribute

The name attribute is used to create a named anchor. When using named anchors we can create links that can jump directly into a specific section on a page, instead of letting the user scroll around to find what he/she is looking for.

Below is the syntax of a named anchor:

```
<a name="label">Text to be displayed</a>
```

The name attribute is used to create a named anchor. The name of the anchor can be any text you care to use.

The line below defines a named anchor:

```
<a name="tips">Useful Tips Section</a>
```

You should notice that a named anchor is not displayed in a special way.

To link directly to the "tips" section, add a # sign and the name of the anchor to the end of a URL, like this:

```
<a href="http://www.w3schools.com/html_links.asp#tips">
Jump to the Useful Tips Section</a>
```

A hyperlink to the Useful Tips Section from WITHIN the file "html_links.asp" will look like this:

```
<a href="#tips">Jump to the Useful Tips Section</a>
```

1.4.13. Frames

With frames, you can display more than one HTML document in the same browser window. Each HTML document is called a frame, and each frame is independent of the others.

The disadvantages of using frames are:

- The web developer must keep track of more HTML documents
- It is difficult to print the entire page

The Frameset Tag

- The <frameset> tag defines how to divide the window into frames
- Each frameset defines a set of rows **or** columns
- The values of the rows/columns indicate the amount of screen area each row/column will occupy

---

The Frame Tag

- The <frame> tag defines what HTML document to put into each frame

In the example below we have a frameset with two columns. The first column is set to 25% of the width of the browser window. The second column is set to 75% of the width of the browser window. The HTML document "frame_a.htm" is put into the first column, and the HTML document "frame_b.htm" is put into the second column:

```
<frameset cols="25%,75%">
  <frame src="frame_a.htm">
  <frame src="frame_b.htm">
</frameset>
```

Note: The frameset column size value can also be set in pixels (cols="200,500"), and one of the columns can be set to use the remaining space (cols="25%,*").

Self Check - Useful Tips

If a frame has visible borders, the user can resize it by dragging the border. To prevent a user from doing this, you can add noresize="noresize" to the <frame> tag.

Add the <noframes> tag for browsers that do not support frames.

Important: You cannot use the <body></body> tags together with the <frameset></frameset> tags! However, if you add a <noframes> tag containing some text for browsers that do not support frames, you will have to enclose the text in <body></body> tags! See how it is done in the first example below.

Frame Tags

Tag	Description
<frameset>	Defines a set of frames
<frame>	Defines a sub window (a frame)
<noframes>	Defines a noframe section for browsers that do not handle frames
<iframe>	Defines an inline sub window (frame)

1.4.14. Tables

Tables are defined with the <table> tag. A table is divided into rows (with the <tr> tag), and each row is divided into data cells (with the <td> tag). The letters td stands for "table data," which is the content of a data cell. A data cell can contain text, images, lists, paragraphs, forms, horizontal rules, tables, etc.

```
<table border="1">
<tr>
```

```
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>row 2, cell 2</td>
</tr>
</table>
```

How it looks in a browser:

Row 1, cell 1	row 1, cell 2
Row 2, cell 1	row 2, cell 2

Tables and the Border Attribute

If you do not specify a border attribute the table will be displayed without any borders. Sometimes this can be useful, but most of the time, you want the borders to show.

To display a table with borders, you will have to use the border attribute:

```
<table border="1">
<tr>
<td>Row 1, cell 1</td>
<td>Row 1, cell 2</td>
</tr>
</table>
```

Headings in a Table

Headings in a table are defined with the <th> tag.

```
<table border="1">
<tr>
<th>Heading</th>
<th>Another Heading</th>
</tr>
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>row 2, cell 2</td>
</tr>
</table>
```

How it looks in a browser:

Heading	Another Heading
row 1, cell 1	row 1, cell 2
row 2, cell 1	row 2, cell 2

Empty Cells in a Table

Table cells with no content are not displayed very well in most browsers.

```
<table border="1">
```

```
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td></td>
</tr>
</table>
```

How it looks in a browser:

row 1, cell 1	row 1, cell 2
row 2, cell 1	

Note that the borders around the empty table cell are missing (NB! Mozilla Firefox displays the border).

To avoid this, add a non-breaking space ( ) to empty data cells, to make the borders visible:

```
<table border="1">
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td> </td>
</tr>
</table>
```

How it looks in a browser:

row 1, cell 1	row 1, cell 2
row 2, cell 1	

Basic Notes - Useful Tips

The <thead>,<tbody> and <tfoot> elements are seldom used, because of bad browser support. Expect this to change in future versions of XHTML. If you have Internet Explorer 5.0 or newer, you can view a working example in our XML tutorial.

Self Check

Table with no border
This example demonstrates a table with no borders.

Headings in a table
This example demonstrates how to display table headers.

Empty cells
This example demonstrates how to use " " to handle cells that have no content.

Table with a caption
This example demonstrates a table with a caption.

Table cells that span more than one row/column

This example demonstrates how to define table cells that span more than one row or one column.

Tags inside a table

This example demonstrates how to display elements inside other elements.

Cell padding

This example demonstrates how to use cellpadding to create more white space between the cell content and its borders.

Cell spacing

This example demonstrates how to use cellspacing to increase the distance between the cells.

Add a background color or a background image to a table

This example demonstrates how to add a background to a table.

Add a background color or a background image to a table cell

This example demonstrates how to add a background to one or more table cells.

Align the content in a table cell

This example demonstrates how to use the "align" attribute to align the content of cells, to create a "nice-looking" table.

The frame attribute

This example demonstrates how to use the "frame" attribute to control the borders around the table.

The frame and border attributes

How to use the "frame" and "border" attributes to control the borders around the table.

Table Tags

Tag	Description
<table>	Defines a table
<th>	Defines a table header
<tr>	Defines a table row
<td>	Defines a table cell
<caption>	Defines a table caption
<colgroup>	Defines groups of table columns
<col>	Defines the attribute values for one or more columns in a table
<thead>	Defines a table head
<tbody>	Defines a table body
<tfoot>	Defines a table footer

1.4.15.Unordered Lists

An unordered list is a list of items. The list items are marked with bullets (typically small black circles).

An unordered list starts with the <ul> tag. Each list item starts with the <li> tag.

```
<ul>
<li>Coffee</li>
<li>Milk</li>
</ul>
```

Here is how it looks in a browser:

- Coffee
- Milk

Inside a list item you can put paragraphs, line breaks, images, links, other lists, etc.

Ordered Lists

An ordered list is also a list of items. The list items are marked with numbers.

An ordered list starts with the <ol> tag. Each list item starts with the <li> tag.

```
<ol>
<li>Coffee</li>
<li>Milk</li>
</ol>
```

Here is how it looks in a browser:

1. Coffee
2. Milk

Inside a list item you can put paragraphs, line breaks, images, links, other lists, etc.

Definition Lists

A definition list is **not** a list of items. This is a list of terms and explanation of the terms.

A definition list starts with the <dl> tag. Each definition-list term starts with the <dt> tag. Each definition-list definition starts with the <dd> tag.

```
<dl>
<dt>Coffee</dt>
<dd>Black hot drink</dd>
<dt>Milk</dt>
<dd>White cold drink</dd>
</dl>
```

Here is how it looks in a browser:

Coffee

Black hot drink

Milk

White cold drink

Inside a definition-list definition (the <dd> tag) you can put paragraphs, line breaks, images, links, other lists, etc.

Self Check

Different types of ordered lists

This example demonstrates different types of ordered lists.

Different types of unordered Lists

This example demonstrates different types of unordered lists.

Nested list

This example demonstrates how you can nest lists.

Nested list 2

This example demonstrates a more complicated nested list.

Definition list

This example demonstrates a definition list.

List Tags

Tag	**Description**
	Defines an ordered list
	Defines an unordered list
	Defines a list item
<dl>	Defines a definition list
<dt>	Defines a definition term
<dd>	Defines a definition description
<dir>	Deprecated. Use instead
<menu>	Deprecated. Use instead

Scripting languages such as Perl[9], Python[4], Rexx[6], Tcl[8], Visual Basic, and the Unix shells represent a very different style of programming than system programming languages. Scripting languages assume that there already exists a collection of useful components written in other languages. Scripting languages aren't intended for writing applications from scratch; they are intended primarily for plugging together components. For example, Tcl and Visual Basic can be used to arrange collections of user interface controls on the screen, and Unix shell scripts are used to assemble filter programs into

pipelines. Scripting languages are often used to extend the features of components but they are rarely used for complex algorithms and data structures; features like these are usually provided by the components. Scripting languages are sometimes referred to as glue languages or system integration languages.

In order to simplify the task of connecting components, scripting languages tend to be typeless: all things look and behave the same so that they are interchangeable. For example, in Tcl or Visual Basic a variable can hold a string one moment and an integer the next. Code and data are often interchangeable, so that a program can write another program and then execute it on the fly. Scripting languages are often string-oriented, since this provides a uniform representation for many different things.

A typeless language makes it much easier to hook together components. There are no a priori restrictions on how things can be used, and all components and values are represented in a uniform fashion. Thus any component or value can be used in any situation; components designed for one purpose can be used for totally different purposes never foreseen by the designer. For example, in the Unix shells, all filter programs read a stream of bytes from an input and write a string of bytes to an output; any two programs can be connected together by attaching the output of one program to the input of the other. The following shell command stacks three filters together to count the number of lines in the selection that contain the word "scripting":

select | grep scripting | wc
The select program reads the text that is currently selected on the display and prints it on its output; the grep program reads its input and prints on its output the lines containing

"scripting"; the wc program counts the number of lines on its input. Each of these programs can be used in numerous other situations to perform different tasks.

The strongly typed nature of system programming languages discourages reuse. Typing encourages programmers to create a variety of incompatible interfaces ("interfaces are good; more interfaces are better"). Each interface requires objects of specific types and

the compiler prevents any other types of objects from being used with the interface, even if that would be useful. In order to use a new object with an existing interface, conversion code must be written to translate between the type of the object and the type expected by the interface. This in turn requires recompiling part or all of the application, which isn't possible in the common case where the application is distributed in binary form.

To see the advantages of a typeless language, consider the following Tcl command:

```
button .b -text Hello! -font {Times 16} -command {puts hello}
```

This command creates a new button control that displays a text string in a 16-point Times font and prints a short message when the user clicks on the control. It mixes six different types of things in a single statement: a command name (button), a button control (.b), property names (-text, -font, and -command), simple strings (Hello! and hello), a font name (Times 16) that includes a typeface name (Times) and a size in points (16), and a Tcl script (puts hello). Tcl represents all of these things uniformly with strings. In this example the properties may be specified in any order and unspecified properties are given default values; more than 20 properties were left unspecified in the example.

The same example requires 7 lines of code in two methods when implemented in Java. With C++ and Microsoft Foundation Classes, it requires about 25 lines of code in three procedures (see [7] for the code for these examples). Just setting the font requires several lines of code in Microsoft Foundation Classes:

```
CFont *fontPtr = new CFont();

fontPtr->CreateFont(16, 0, 0,0,700, 0, 0, 0, ANSI_CHARSET,

   OUT_DEFAULT_PRECIS,CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
```

```
    DEFAULT_PITCH|FF_DONTCARE, "Times New Roman");

buttonPtr->SetFont(fontPtr);
```

Much of this code is a consequence of the strong typing. In order to set the font of a button, its SetFont method must be invoked, but this method must be passed a pointer to a CFont object. This in turn requires a new object to be declared and initialized. In order to initialize the CFont object its CreateFont method must be invoked, but CreateFont has a rigid interface that requires 14 different arguments to be specified. In Tcl, the essential characteristics of the font (typeface Times, size 16 points) can be used immediately with no declarations or conversions. Furthermore, Tcl allows the behavior for the button to be included directly in the command that creates the button, while C++ and Java require it to be placed in a separately declared method.

(In practice, a trivial example like this would probably be handled with a graphical development environment that hides the complexity of the underlying language: the user enters property values in a form and the development environment outputs the code. However, in more complex situations such as conditional assignment of property values or interfaces generated programmatically, the developer must write code in the underlying language.)

It might seem that the typeless nature of scripting languages could allow errors to go undetected, but in practice scripting languages are just as safe as system programming languages. For example, an error will occur if the font size specified for the button example above is a non-integer string such as xyz. The difference is that scripting languages do their error checking at the last possible moment, when a value is used. Strong typing allows errors to be detected at compile-time, so the cost of run-time checks is avoided. However, the price to be paid for this efficiency is restrictions on how information can be used: this results in more code and less flexible programs.

Another key difference between scripting languages and system programming languages is that scripting languages are usually interpreted whereas system programming languages are usually compiled. Interpreted languages provide rapid turnaround during development by eliminating compile times. Interpreters also make applications more flexible by allowing users to program the applications at run-time. For example, many synthesis and analysis tools for integrated circuits include a Tcl interpreter; users of the programs write Tcl scripts to specify their designs and control the operation of the tools. Interpreters also allow powerful effects to be achieved by generating code on the fly. For example, a Tcl-based Web browser can parse a Web page by translating the HTML for the page into a Tcl script using a few regular expression substitutions. It then executes the Tcl script to render the page on the screen.

Scripting languages are less efficient than system programming languages, in part because they use interpreters instead of compilers but also because their basic components are chosen for power and ease of use rather than an efficient mapping onto the underlying hardware. For example, scripting languages often use variable-length strings in situations where a system programming language would use a binary value that fits in a single machine word, and scripting languages often use hash tables where system programming languages use indexed arrays.

Fortunately, the performance of a scripting language isn't usually a major issue. Applications for scripting languages are generally smaller than applications for system programming languages, and the performance of a scripting application tends to be dominated by the performance of the components, which are typically implemented in a system programming language.

Review

Summary

- Web Publishing provides custom web design, web development, hosting, e-commerce, and e-business solutions.
- We work closely with our clients to produce website structure, design, and content that is affordable, functional, attractive, and reflects the spirit of their business.
- A presence on the Web promotes a company's image and products, improves customer service, encourages new customer acquisition, and provides a vehicle for sales and information**.**
- **Client-server** computing or networking is a distributed application architecture that partitions tasks or work loads between service providers (servers) and service requesters, called clients. Often clients and servers operate over a computer network on separate hardware. A server machine is a high-performance host that is running one or more server programs which share its resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await (*listen* to) incoming requests.
- **HTML**, which stands for **Hyper Text Markup Language**, is the predominant markup language for web pages. It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists etc as well as for links, quotes, and other items. It allows images and objects to be embedded and can be used to create interactive forms. It is written in the form of HTML elements consisting of "tags" surrounded by angle brackets within the web page content.

- It can include or can load scripts in languages such as JavaScript which affect the behavior of HTML processors like Web browsers; and Cascading Style Sheets (CSS) to define the appearance and layout of text and other material. The W3C, maintainer of both HTML and CSS standards, encourages the use of CSS over explicit presentational markup.

Key Terms ::

TCP, UDP
Client/server, RPC, LPC
HTML, DHTML, SGML, XHTML, CSS
HTTP, URL, DNS

Key Terms Quiz:

Complete each statement by writing one of the terms listed under key terms in each blank

1. The process of copying a file from a remote computer to your local computer ----------
2. Uploading means----------
3. ---------- is a superset of Markup Language
4. ---------- is the format of DNS
5. Load balancing says ----------
6. HTTPS means ----------.
7. ---------- the default por t number of HTTPS.
8. CSS is used for ----------.
9. ---------- is used to insert a comment in the HTML source code.
10. TCP is used for ----------.

Multiple Choice ::

1. Browsing Means ----------

a. Message tracing b. navigation

c. Hyper link Access d.None of the above

2. HTML was coined by ----------

a. Tim Berners Lee b. Achyat S.Godbole

c.James Gosling d.None of the above

3. An ordered list starts with the.

a. tag b. <o> tag

c. <l>tag d. none of the above

4. PERL is extended as------

a. Practical Extraction and Report Language b. Practical Extension and Report Language c. Practical Extraction and Reengineering Language d.none of the above

5. HTML uses ---------- tags to describe web pages

a. mark up tags b.userdefined tags

c. display tags d. all of the above

6. HTML headings are defined with ----------

a .<h1> to<h6> b.<h1>to <h8>

c. <h1>&<h2> d.<h1>to <h12>

Review Questions::

In your own words briefly answer the following questions

1. What is HTML?

2. What is the extension of HTML or HTML Extension?
3. How do you include Image in HTML?
4. What is the use of <a> tag in HTML?
5. Write down the HTML Rules.
6. How do you make a comment line statement in HTML?.
7. How do you write a paragraph in HTML?
8. What is Client Server computing?
9. What is meant by RPC?
10. What is meant by Load balancing?

Lesson lab

1. To develop a web page to display the details about Chettinad College of Engineering using HTML tags and Hyperlinks.

2. To Create a dynamic web page for Online E- Mail Registration form using HTML