तेजस्वि नावधीतमस्तु
ISO 9001:2008 & 14001:2004

FAIRFIELD
Institute of Management & Technology
Managed by 'The Fairfield Foundation'
(Affiliated to GGSIP University, New Delhi)

CODE:- BCA –204
SUBJECT:- Web Technologies

UNIT – I

History of the Internet and World Wide Web, Search Engines, News-group, E-mail and its Protocols, Web Portal, Browsers and their versions, Its functions, URLs, web sites, Domain names, Portals.

Static Web Development: HTML - Introduction to HTML, HTML Document structure tags, HTML comments, Text formatting, inserting special characters, anchor tag, adding images and Sound, lists types of lists, tables, frames and floating frames, Developing Forms, Image maps.

UNIT – II

Introduction to Java Script: Data Types, Control Statements, operators, Built in and User Defined Functions, Objects in Java Script, Handling Events.

Cascading Style Sheet: Types of Style Sheets – Internal, inline and External style sheets, creating styles, link tag.

UNIT – III

DHTML: Introduction to DHTML, JavaScript & DHTML, Document Object Model, Filters and Transitions, DHTML Events, Dynamically change style to HTML Documents.

UNIT – IV

Introduction to WYSIWYG Design tools, Introduction to Dreamweaver, Website Creation and maintenance, Web Hosting and Publishing Concepts, XML: Introduction to XML-Mark up languages, Features of Mark up languages, XML Naming rules, Building block of XML Document, Difference between HTML & XML

Components of XML, XML Parser, DTD's Using XML with HTML and CSS

Unit –I

History of the Internet and World Wide Web

Before there was the public internet there was the internet's forerunner ARPAnet or Advanced Research Projects Agency Networks. ARPAnet was funded by the United States military after the cold war with the aim of having a military command and control center that could withstand nuclear attack. The point was to distribute information between geographically dispersed computers. ARPAnet created the TCP/IP communications standard, which defines data transfer on the Internet today. The ARPAnet opened in 1969 and was quickly usurped by civilian computer nerds who had now found a way to share the few great computers that existed at that time.

The **World Wide Web** is a global information medium which users can read and write via computers connected to the Internet. The term is often mistakenly used as a synonym for the Internet itself, but the Web is a service that operates over the Internet, just as e-mail also does. The history of the Internet dates back significantly further than that of the World Wide Web.

Search Engines

A **web search engine** is a software system that is designed to search for information on the World Wide Web. The search results are generally presented in a line of results often referred to as search engine results pages (SERPs). The information may be a specialist in web pages, images, information and other types of files. Some search engines also mine data available in databases or open directories. Unlike web directories, which are maintained only by human editors, search engines also maintain real-time information by running an algorithm on a web crawler.

A search engine operates in the following order:

1. Web crawling
2. Indexing
3. Searching

Web search engines work by storing information about many web pages, which they retrieve from the page's HTML. These pages are retrieved by a Web crawler (sometimes also known as a spider) — an automated Web browser which follows every link on the site.

News- group

A **Usenet newsgroup** is a repository usually within the Usenet system, for messages posted from many users in different locations. The term may be confusing to some, because it is in fact a discussion group. Newsgroups are technically distinct from, but functionally similar to, discussion forums on the World Wide Web. Newsreader software is used to read newsgroups.

Newsgroups generally come in either of two types, binary or text. There is no technical difference between the two, but the naming differentiation allows users and servers with limited facilities to minimize network bandwidth usage.

Email and its Protocols

Electronic mail, most commonly referred to as **email** or **e-mail** since approximately 1993, is a method of exchanging digital messages from an author to one or more recipients. Modern email operates across the Internet or other computer networks. Some early email systems required that the author and the recipient both be online at the same time, in common with instant messaging.

An Internet email message consists of three components, the message *envelope*, the message *header*, and the message *body*.

Web-based email (webmail)

This is the type of email that most users are familiar with. Many free email providers host their serves as web-based email (e.g. Hotmail, Yahoo, Gmail, and AOL).

POP3 email services

POP3 is the acronym for Post Office Protocol 3. It is a leading email account type on the Internet. In a POP3 email account, email messages are downloaded to the client device and then they are deleted from the mail server. It is difficult to save and view messages on multiple devices. Also, the messages sent from the computer are not copied to the Sent Items folder on the devices.

IMAP email servers

IMAP refers to Internet Message Access Protocol. It is an alternative to the POP3 email. With an IMAP account, a user's account has access to mail folders on the mail server and can use any compatible device to read messages, as long as such a device can access the server.

MAPI email servers

Messaging Application Programming Interface (MAPI) is a messaging architecture and a Component Object Model based API for Microsoft Windows.

Web Portal

A **web portal** is a website that brings information together from diverse sources in a uniform way. Usually, each information source gets its dedicated area on the page for displaying information often; the user can configure which ones to display.

Browsers and their versions

The increased growth of the Internet in the 1990s and 2000s means that current browsers with small market shares have more total users than the entire market early on. For example, 90% market share in 1997 would be roughly 60 million users, but by the start of 2007 9% market share would equate to over 90 million users.

- WorldWideWeb, February 26, 1991
- Mosaic, April 22, 1993
- Netscape Navigator and Netscape Communicator, October 13, 1994
- Internet Explorer, August 16, 1995
- Opera, 1996
- Mozilla Navigator, June 5, 2002
- Safari, January 7, 2003
- Mozilla Firefox, November 9, 2004
- Google Chrome, September 2, 2008

Browser's functions i.e, Its functions

The primary purpose of a web browser is to bring information resources to the user ("retrieval" or "fetching"), allowing them to view the information ("display", "rendering"), and then access other information ("navigation", "following links").

This process begins when the user inputs a Uniform Resource Locator (URL), for example *http://en.wikipedia.org/*, into the browser. The prefix of the URL, the Uniform Resource Identifier or URI, determines how the URL will be interpreted. The most commonly used kind of URI starts with *http:* and identifies a resource to be Retrieved over the Hypertext Transfer Protocol (HTTP).[10] Many browsers also support a variety of other prefixes, such as *https:* for HTTPS, *ftp:* for the File Transfer Protocol, and *file:* for local files. Prefixes that the web browser cannot directly handle are often handed off to another application entirely. For example, *mailto:* URIs are usually passed to the user's default e-mail application, and *news:* URIs are passed to the user's default newsgroup reader.

In the case of *http*, *https*, *file*, and others, once the resource has been Retrieved the web browser will display it. HTML and associated content (image files, formatting information such as CSS, etc.) is passed to the browser's layout engine to be transformed from markup to an interactive document, a process known as "rendering". Aside from HTML, web browsers can generally display any kind of content that can be part of a web page. Most browsers can display images, audio, video, and XML files, and often have plug-ins to support Flash applications and Java applets. Upon encountering a file of an unsupported type or a file that is set up to be downloaded rather than displayed, the browser prompts the user to save the file to disk.

Information resources may contain hyperlinks to other information resources. Each link contains the URI of a resource to go to. When a link is clicked, the browser navigates to the resource indicated by the link's target URI, and the process of bringing content to the user begins again.

URLs

A **uniform resource locator** also known as **web address** is a specific character string that constitutes a reference to a resource. In most web browsers, the URL of a web page is displayed on top inside an address bar.

Web sites

A **website**, also written as **Web site**, **web site**, or simply **site**, is a set of related web pages served from a single web domain. A website is hosted on at least one web server, accessible via a network such as the Internet or a private local area network through an Internet address known as a Uniform Resource Locator. All publicly accessible websites collectively constitute the World Wide Web.

Domain names

A **domain name** is an identification string that defines a realm of administrative autonomy, authority, or control on the Internet. Domain names are formed by the rules and procedures of the Domain Name System (DNS). Technically, any name registered in the DNS is a domain name.

Domain names are used in various networking contexts and application-specific naming and addressing purposes. In general, a domain name represents an Internet Protocol (IP) resource, such as a personal computer used to access the Internet, a server computer hosting a web site, or the web site itself or any other service communicated via the Internet.

Portals

- Enterprise portal, a framework to provide a single point of access to a variety of information and tools
- Intranet portal, a gateway that unifies access to all enterprise information and applications
- Portal rendering, an optimization technique in 3D computer graphics
- Web portal, a site that functions as a point of access to information on the Internet

Static Web Development: HTML

A leading **web design** company, we combine knowledge, experience and talent to produce cutting edge visuals in web designing. Our **Static website** packages provide absolute solution to the businesses or individuals, to post simple information about themselves or about their company onto the **static web pages.**

Our professional website designers work to produce stunning imagery, meaningful content and user-friendly ecommerce applications.
Creates an attractive professional website which is easy to navigate.
Provides static web content with relevant keywords which gets your website top ranking in major search engines like Google and Yahoo etc.
Static website design thus strikes a balance between good-quality images and fast downloads time.

We give unique and personalized **web page design** with quality at competitive market rates.

To generate good number of business inquiries, you will get: Your website would be promoted on all top search engines and directory like Google, Yahoo.

It Includes :

Domain Renewal for one year
Space Renewal (MB) for one year.
Minor updating at the website.

Introduction to HTML

Several technologies (such as CSS, JavaScript, Flash, AJAX, JSON) can be used to define the elements of a web page. However, at the very lowest level, a web page is defined using **HTML** (**HyperText Markup Language**). Without HTML, there is no web page.

What is HTML?

HTML is a **markup language**. It tells the web browser how to display content. HTML separates "content" (words, images, audio, video, and so on) from "presentation" (the definition of the type of content and the instructions for how that type of content should be displayed). HTML uses a pre-defined set of elements to identify content types.

For example, the paragraph element consists of the start tag "<p>" and the closing tag "</p>". The following example show a paragraph contained within the HTML paragraph element:

```
<p>My dog ate all the guacamole.</p>
```

When this content is displayed in a web browser, it looks like this:

The browser uses the tags as an indicator of how to display the content in the tags.

HTML Document structure tags

HTML (Outermost tag):- The HTML tag identifies a document as an HTML document. All HTML documents should start with the <HTML> tag and end with the </HTML> tag.

Syntax

<HTML>...</HTML>

HEAD (Document header)

The HEAD tag defines an HTML document header. The header contains information about the document rather than information to be displayed in the document. The web browser displays none of the information in the header, except for text contained by the TITLE tag. You should put all header information between the <HEAD> and </HEAD> tags, which should precede the BODY tag.

Syntax

<HEAD>...</HEAD>

TITLE (Document header)

The TITLE tag defines the TITLE of the document. This is what is displayed in the top of your browser window. In addition, many search engines use this as their primary name of a document.

Syntax

<TITLE>...</TITLE>

BODY (Main content of document)

The BODY tag specifies the main content of a document. You should put all content that is to appear in the web page between the <BODY> and </BODY> tags.

Syntax

<BODY ...</BODY>

HTML Comments

<!--This is a comment. Comments are not displayed in the browser-->

<p>This is a paragraph. </p>

Text formatting

Tag	**Description**
<b>	Defines bold text
<em>	Defines emphasized text
<i>	Defines a part of text in an alternate voice or mood
<small>	Defines smaller text
<strong>	Defines important text
<sub>	Defines subscripted text
<sup>	Defines superscripted text
<ins>	Defines inserted text
<del>	Defines deleted text

Inserting special characters

You can use the Symbol dialog box to insert symbols, such as ¼ and ©, or special characters, such as an em dash (—) or ellipsis (…) that are not on your keyboard, as well as Characters.

Anchor tags

The anchor tag will consist of two HTML elements. First, you'll want to create the link.

If you are linking to a spot on the same page, the format of the link will be similar to: <a href="#anchor">Link Text</a>

For example, if the text is "Read more about raptors!" then your HTML should look like this:

<a href="#raptors">Read more about raptors! </a>

Adding images and sound

In order to link a sound to a picture, follow the steps in the video, as outlined here:

1. add a picture from your gallery or by using the screen clipping tool
2. Right click on the picture and select "hyperlink"
3. Change the feature to "play sound"
4. Click "load" and find the sound (wherever you have it saved on your computer) OR if it's one in the gallery, find it in the gallery and drag it to the box on this little pop up screen
5. Click "ok." Now the little orange play symbol should appear when you hover over the image

To add one of the little speakers instead of linking the sound to a picture:

1. along the top of the page, select "insert - file"
2. Find the sound file wherever it is saved on your computer and click "open"
3. The little speaker should appear on the page and can be moved or resized as needed

Lists types of lists

HTML Unordered Lists

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

The list items are marked with bullets (typically small black circles).

HTML Ordered Lists

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

The list items are marked with numbers.

HTML Description Lists

A description list is a list of terms/names, with a description of each term/name.

The <dl> tag defines a description list.

The <dl> tag is used in conjunction with <dt> (defines terms/names) and <dd> (describes each term/name):

Table: A table is a collection of rows & columns.

```
<table border="1">
            <tr>
        <th>Month</th>
```

```
<th>Savings</th>
</tr>
<tr>
<td>January</td>
<td>$100</td>
</tr>
</table>
```

Frames & floating frames

HTML stands for hyper text markup language which is generally used to create web pages. While creating web pages formation of frames is also a vital part to know. Basically a framed document divides a browser window into multiple panes or smaller window frames. Each frame can contain a different document. The benefit of making frames are user can view information in one frame while keeping another frame open for reference instead of moving back and forth between pages. The contents of one frame can be manipulated or linked to the contents of other. This enables the web page designer to build sophisticated interfaces.

Developing Forms, Image maps

If you've ever wanted to create image maps for your web pages, you can do so in a variety of ways. One approach is to do so manually, though that could be quite time-consuming. A better way is to make use of software to help you, whether that's a standalone application or an option that's a part of a larger application.

An image map could be a simple as a grid with clickable hot spots or it could be as complicated as setting regions on a map, which would make use of multiple shapes. Each clickable section will then take you to a different web page.

UNIT-II (Introduction to Java Script)

Data types in Java Script

JavaScript has dynamic types. This means that the same variable can be used as different types:

Example

var x; =// Now x is undefined
var x = 5; // Now x is a Number
var x = "John"; // Now x is a String

JavaScript Strings

A string is a variable which stores a series of characters like "John Doe".

A string can be any text inside quotes. You can use single or double quotes:

Example

```
Var carname="Volvo XC60";
var carname='Volvo XC60';
```

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

Example

```
var answer="It's alright";
var answer="He is called 'Johnny'";
var answer='He is called "Johnny"';
```

JavaScript Numbers

JavaScript has only one type of numbers. Numbers can be written with, or without decimals:

Example

```
var x1=34.00;    var x2=34;
```

Extra large or extra small numbers can be written with scientific (exponential) notation:

Example

```
var y=123e5;    // 12300000
var z=123e-5;   // 0.00123
```

JavaScript Booleans

Booleans can only have two values: true or false.

JavaScript Arrays

The following code creates an Array called cars:

```
var cars=new Array();
cars[0]="Saab";
cars[1]="Volvo";
cars[2]="BMW";
```

Or (condensed array):

```
var cars=new Array("Saab","Volvo","BMW");
```

Or (literal array):

Example

var cars= ["Saab","Volvo","BMW"];

Control Statements

This page describes the use of JavaScript control statements such as the if statement, while statement, the for statement, and the switch statement. It provides examples.

If

```
if (value == 3)
{
  document.write("Value is three")
}
else
{
  document.write("Value is not three")
}
```

While

```
var i = 0
while (i < 3)
{
  i++
}
```

Do While

The example prints the days of the week.

```
days = new Array("Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday")
var day = 0
do
{
  document.write(days[day] + " is day " + eval(day+1) + " of the week.<BR>\n")
  day++
} while (day < 7)
```

For

The following example will determine what numeric day of the week Wednesday falls on.

```
days = new
Array("Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday
")
var day = "Wednesday"
var place
for (i = 0; i < days.length; i++)
{
  if (day == days[i])
  {
    place = i + 1
  }
}
document.write(day + " is day " + place + " in the week.<BR>\n")
```

The switch statement

Runs one or more commands based on the value of the expression being evaluated. An example is:

```
switch (day) {
  case 1:
    document.write("Sunday<BR>\n")
    break
  case 2:
    document.write("Monday<BR>\n")
    break
  case 3:
    document.write ("Tuesday<BR>\n")
    break
  case 4:
    document.write ("Wednesday<BR>\n")
    break
  case 5:
    document.write ("Thursday<BR>\n")
    break
  case 6:
    document.write ("Friday<BR>\n")
    break
  case 7:
    document.write ("Saturday<BR>\n")
    break
  default:
   document.write ("Invalid Weekday<BR>\n")
   break
}
```

Break

The break statement may be used to break out of a while, if, switch, do while, or for statement. The break causes program control to fall to the statement following the iterative statement. The break statement now supports a label which allows the specification of a location for the program control to jump to. The following code will print "Wednesday is day 4 in the week".

```
days = new
Array("Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday
")
var day = "Wednesday"
var place
for (i = 0;i < days.length;i++)
{
  if (day == days[i]) break;
}
place = i + 1
document.write(day + " is day " + place + " in the week.<BR>\n")
```

Continue

The continue statement does not terminate the loop statement, but allows statements below the continue statement to be skipped, while the control remains with the looping statement. The loop is not exited. The example below does not include any ratings values below 5.

```
ratings = new Array(6,9,8,4,5,7,8,10)
var sum = 0;
var reviews = 0
for (i=0; i < ratings.length; i++)
{
  if (ratings[i] < 5) continue
  sum += ratings[i];
  reviews++;
}
var average = sum/reviews
```

For in Statement

Executes a loop similar to a for statement that executes for all the properties in an object. The example below lists all properties and their values for the document object. The document is the HTML page being displayed.

```
for (i in document)
{
        document.write("Property = " + i + " Value = " + document[i] + "<BR>\n")
    }
```

Operators in Java Script

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic between variables and/or values.

Operator	Description	Example	Result of x	Result of y
+	Addition	x=y+2	7	5
-	Subtraction	x=y-2	3	5
*	Multiplication	x=y*2	10	5
/	Division	x=y/2	2.5	5
%	Modulus (division remainder)	x=y%2	1	5
++	Increment	x=++y	6	6
		x=y++	5	6
--	Decrement	x=--y	4	4
		x=y--	5	4

JavaScript Assignment Operators

Assignment operators are used to assign values to JavaScript variables.

Given that **x=10** and **y=5**, the table below explains the assignment operators:

Operator	Example	Same As	Result
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5

*=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

Built in & user defined functions

A function is a group of related JavaScript statements that exist separately from the rest of the script and perform one or more designated tasks. Functions are also useful if you need to control exactly when a particular task occurs. The basic structure of a function is a follows:

```
function Function_Name([Arguments]) {
   JavaScript statements
}
```

The **function** keyword identifies the procedure as a function. **Function Name** this is a unique name for the function. Example: Calling a function:

```
<html>
<head>
<title>Calling a Function</title>
<script language="JavaScript" type="text/JavaScript">

function make_background_red() {
   document.bgColor = "red"
   document.write ("<h1>The background is now red.<\/h1>")
}
make_background_red()
</script>
</head>
<body>
</body>
</html>
```

Objects in Java Script

In JavaScript, objects are data (variables), with properties and methods.

The String object has a built-in property called length. For the string above, length has the value 5. The String objects also have several built-in methods.

Handling Events

Events occur when some sort of interaction takes place in a web page. This can be the end user clicking on something, moving the mouse over a certain element or pressing down certain keys on the keyboard. An event can also be something that happens in the web browser, such as the web page completing the loading of a page, or the user scrolling or resizing the window.

There are two types of event order: *event capturing* and *event bubbling*.

Event capturing starts with the outer most elements in the DOM and works inwards to the HTML element the event took place on and then out again. For example, a click in a web page would first check the HTML element for onclick event handlers, then the body element, and so on, until it reaches the target of the event.

Event bubbling works in exactly the opposite manner: it begins by checking the target of the event for any attached event handlers, and then bubbles up through each respective parent element until it reaches the HTML element.

The evolution of events

In the early days of Java Scripting, we used event handlers directly within the HTML element, like this:

```
<a href="http://www.opera.com/" onclick="alert ('Hello')">Say hello</a>
```

The problem with this approach is that it resulted in event handlers spread throughout the code, no central control and missing out on web browsers' caching features when it comes to external JavaScript file includes.

The next step in event evolution was to apply events from within a JavaScript block, for example:

```
<script type="text/javascript">
  document.getElementById("my-link").onclick = waveToAudience;
    function waveToAudience() {
      alert ("Waving like I've never waved before!");
    }
</script>
```

```
<a id="my-link" href="http://www.opera.com/">My link</a>
```

Cascading Style Sheet: Types of Style Sheets- Internal, inline and external style sheets, creating styles, link tag.

Style sheets represent a major breakthrough for Web page designers, expanding their ability to improve the appearance of their pages. In the scientific environments in which the Web was conceived, people are more concerned with the content of their documents than the presentation. They include:

- Using proprietary HTML extensions
- Converting text into images
- Using images for white space control
- Use of tables for page layout
- Writing a program instead of using HTML

These techniques considerably increase the complexity of Web pages, offer limited flexibility, suffer from interoperability problems, and create hardships for people with disabilities.

Style sheets solve these problems at the same time they supersede the limited range of presentation mechanisms in HTML. Style sheets make it easy to specify the amount of white space between text lines, the amount lines are indented, the colors used for the text and the backgrounds, the font size and style, and a host of other details.

For example, the following short CSS style sheet (stored in the file "special.css"), sets the text color of a paragraph to green and surrounds it with a solid red border:

```
P.special {
color: green;
border: solid red;
}
```

Authors may link this style sheet to their source HTML document with the LINK element:

```
<HTML>
  <HEAD>
    <LINK href="special.css" rel="stylesheet" type="text/css">
  </HEAD>
  <BODY>
    <P class="special">This paragraph should have special green text.
  </BODY>
</HTML>
```

HTML 4 provides support for the following style sheet features:

Flexible placement of style information

Placing style sheets in separate files makes them easy to reuse. Sometimes it's useful to include rendering instructions within the document to which they apply, either grouped at the start of the document, or in attributes of the elements throughout the body of the document.

Independence from specific style sheet languages

This specification doesn't tie HTML to any particular style sheet language. This allows for a range of such languages to be used, for instance simple ones for the majority of users and much more complex ones for the minority of users with highly specialized needs.

Cascading

This is the capability provided by some style sheet languages such as CSS to allow style information from several sources to be blended together. These could be, for instance, corporate style guidelines, styles common to a group of documents, and styles specific to a single document.

Media dependencies

HTML allows authors to specify documents in a media-independent way. This allows users to access Web pages using a wide variety of devices and media, e.g., graphical displays for computers running Windows, Macintosh OS, and X11.

Alternate styles

Authors may wish to offer readers several ways to view a document. For instance, a style sheet for rendering compact documents with small fonts or one that specifies larger fonts for increased legibility.

Performance concerns

Some people have voiced concerns over performance issues for style sheets. For instance, retrieving an external style sheet may delay the full presentation for the user. A similar situation arises if the document head includes a lengthy set of style rules.

HTML documents may contain style sheet rules directly in them or they may import style sheets.

Inline style sheets

This attribute specifies style information for the current element.

The syntax of the value of the style attribute is determined by the default style sheet language. For example, for inline style, use the declaration block syntax. This CSS example sets color and font size information for the text in a specific paragraph.

```
<P style="font-size: 12pt; color: fuchsia">aren’t style sheets wonderful?
```

```
<HEAD>
 <STYLE type="text/css">
   H1 {border-width: 1; border: solid; text-align: center}
 </STYLE>
</HEAD>
```

To specify that this style information should only apply to H1 elements of a specific class, we modify it as follows:

```
<HEAD>
 <STYLE type="text/css">
   H1.myclass {border-width: 1; border: solid; text-align: center}
 </STYLE>
</HEAD>
<BODY>
 <H1 class="myclass"> This H1 is affected by our style </H1>
 <H1> This one is not affected by our style </H1>
</BODY>
```

Finally, to limit the scope of the style information to a single instance of H1, set the id attribute:

```
<HEAD>
 <STYLE type="text/css">
   #myid {border-width: 1; border: solid; text-align: center}
 </STYLE>
</HEAD>
<BODY>
 <H1 class="myclass"> This H1 is not affected </H1>
 <H1 id="myid"> This H1 is affected by style </H1>
 <H1> This H1 is not affected </H1>
</BODY>
```

External style sheets

Authors may separate style sheets from HTML documents. This offers several benefits:

- Authors and Web site managers may share style sheets across a number of documents (and sites).
- Authors may change the style sheet without requiring modifications to the document.
- User agents may load style sheets selectively.

User agents should provide a means for users to view and pick from the list of alternate styles. The value of the title attribute is recommended as the name of each choice.

In this example, we first specify a persistent style sheet located in the file mystyle.css:

```
<LINK href="mystyle.css" rel="stylesheet" type="text/css">
```

Setting the title attribute makes this the author's preferred style sheet:

```
 <LINK href="mystyle.css" title="compact" rel="stylesheet" type="text/css">
```

Adding the keyword "alternate" to the rel attribute makes it an alternate style sheet:

```
<LINK href="mystyle.css" title="Medium" rel="alternate stylesheet" type="text/css">
```

Cascading style sheets

Cascading style sheet languages such as CSS allow style information from several sources to be blended together. However, not all style sheet languages support cascading. To define a cascade, authors specify a sequence of LINK and/or STYLE elements. The style information is cascaded in the order the elements appear in the HEAD.

```
<LINK rel="alternate stylesheet" title="compact" href="small-base.css" type="text/css">
<LINK rel="alternate stylesheet" title="compact" href="small-extras.css" type="text/css">
<LINK rel="alternate stylesheet" title="big print" href="bigprint.css" type="text/css">
<LINK rel="stylesheet" href="common.css" type="text/css">
```

Here is a cascade example that involves both the LINK and STYLE elements.

```
<LINK rel="stylesheet" href="corporate.css" type="text/css">
<LINK rel="stylesheet" href="techreport.css" type="text/css">
<STYLE type="text/css">
```

Internal Stylesheet

An internal style sheet should be used when a single document has a unique style. You define internal styles in the head section of an HTML page, by using the <style> tag, like this:

```
<head>
<style>
hr {color:sienna;}
p {margin-left:20px;}
body {background-image:url("images/back40.gif");}
</style>
</head>
```

UNIT-III (DHTML)

Introduction to DHTML

Dynamic HTML, or **DHTML**, is an umbrella term for a collection of technologies used together to create interactive and animated web sites by using a combination of a static markup language, a client-side scripting language, a presentation definition language (such as CSS), and the Document Object Model.

DHTML allows scripting languages to change variables in a web page's definition language, which in turn affects the look and function of otherwise "static" HTML page content, *after* the page has been fully loaded and during the viewing process. Thus the dynamic characteristic of DHTML is the way it functions while a page is viewed, not in its ability to generate a unique page with each page load.

JavaScript & DHTML

On numerous online forums for JavaScript and DHTML. After reading thousands of forum threads over the years, author and scripting pioneer Danny Goodman has compiled a list of problems that frequently vex scripter's of various experience levels. He has now applied state-of-the-art ECMA and W3C DOM standards and used best practices to create this extensive collection of practical recipes that can bring your web pages to life.

The *JavaScript & DHTML* is all about adding value to the content of a web page. For every problem Goodman addresses, there's a solution or "recipe"--a focused piece of code that web developers can insert directly into their applications. Yet, rather than just cut-and-paste code, you also get explanations of how and why the code works, so you can learn to adapt the problem-solving techniques to your designs.

The recipes range from simple tasks, such as manipulating strings and validating dates in JavaScript, to entire libraries that demonstrate complex tasks, such as cross-browser positioning of HTML elements and sorting tables. This book contains over 150 recipes on the following topics:

- Working with interactive forms and style sheets
- Presenting user-friendly page navigation
- Creating dynamic content
- Producing visual effects for stationary content
- Positioning HTML elements
- Managing browser windows and multiple frames

Document Object model

The **Document Object Model** (**DOM**) is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML and XML documents. Objects in the DOM tree may be addressed and manipulated by using methods on the objects. The public interface of a DOM is specified in its application programming interface (API). The history of the Document Object Model is intertwined with the history of the "browser wars" of the late 1990s between Netscape Navigator and Microsoft Internet Explorer, as well as with that of JavaScript and JScript, the first scripting languages to be widely implemented in the layout engines of web browsers.

JavaScript was released by Netscape Communications in 1996 within Netscape Navigator 2.0. Netscape's competitor, Microsoft, released Internet Explorer 3.0 later the same year with a port of JavaScript called JScript. JavaScript and JScript let web developers create web pages with client-side interactivity. The limited facilities for detecting user-generated events and modifying the HTML document in the first generation of these languages eventually became known as "DOM Level 0" or "Legacy DOM." No independent standard was developed for DOM Level 0, but it was partly described in the specification of HTML 4.

Legacy DOM was limited in the kinds of elements that could be accessed. Form, link and image elements could be referenced with a hierarchical name that began with the root document object. A hierarchical name could make use of either the names or the sequential index of the traversed elements. For example, a form input element could be accessed as either "document.formName.inputName" or "document.forms[0].elements[0]."

The Legacy DOM enabled client-side form validation and the popular "rollover" effect.

Filters & Transitions

Visual filters are extensions to CSS properties that change the display of an object's contents. In some cases, filters create simple behaviors that could be implemented as script. In other cases, a filter goes far beyond script and modifies the rendered appearance of the content of an object. Using filters simplifies the task of incorporating sophisticated effects in Web documents. Static visual filters affect the display of content, while transitions affect the way content changes in an object are displayed.

For a list of Internet Explorer 5.5 filters and transitions, see the To see the Filters and Transitions Interactive Demo, which demonstrates most of the filters and transitions, click the following button.

Filters are applied to HTML controls through the filter property. The **filter** property is a string of filter descriptions that uses a function-like notation, but you do not need to know script to implement this property. The following syntax shows an Internet Explorer 5.5 **filter** declaration in a STYLE attribute.

```
<ELEMENT STYLE="filter: progid: DXImageTransform.Microsoft.filtername (sProperties)" >
```

The following example shows a **filter** declaration composed of two filters.

```
<IMG ID=sample SRC="sample.jpg"

STYLE="filter: progid:DXImageTransform.Microsoft.MotionBlur(strength=50)
      progid:DXImageTransform.Microsoft.BasicImage(rotation=2, mirror=1); width=50%">
```

This example HTML attaches two filters to the object. The first causes the image to blur over a specified number of pixels. The second filter flips the image vertically.

You can apply multiple filters, as shown in the example.

Filters and transitions display properly only on systems that have the color palette set to display 256 colors or more.

Almost any object can have filters applied to it. However, the object that the filter is applied to must have layout before the filter effect will display. Put simply, *having layout* means that an object has a defined height and width. Some objects, like form controls, have layout by default. All other filterable objects gain layout by setting the height or width property, setting the position property to absolute, setting the writingMode property to tb-rl, or setting the content Editable property to true.

DHTML Events

Clicking a button, moving the mouse pointer over part of the Web page, selecting some text on the page — these actions all fire events and a DHTML author can write code to run in response to the event. This particular piece of code is generally known as an event handler. The Internet Explorer event model makes it possible for every HTML element on the page to be the source for a full set of mouse and keyboard events.

The following tables show a set of common events that every HTML element generates in Internet Explorer.

Mouse events

Mouse event generated	User action
onmouseover	Moves the mouse pointer over (enters) an element.
onmouseout	Moves the mouse pointer away from (exits) an element.
onmousedown	Presses any of the mouse buttons.
onmouseup	Releases any of the mouse buttons.
onmousemove	Moves the mouse pointer within an element.
onclick	Clicks the left mouse button on an element.
ondblclick	Double-clicks the left mouse button on an element.

Keyboard events

Keyboard event generated	User action
onkeypress	Presses and releases a key (full down-and-up cycle). However, if a key is held down, multiple **onkeypress** events are fired.
onkeydown	Presses a key. Only a single event is fired, even if the key is held down.
onkeyup	Releases a key.

For more information about event handling in Internet Explorer, including event bubbling and handling rollover effects

Dynamically change style to HTML Documents

Dynamic HTML (DHTML) is a set of innovative features originally introduced in Microsoft Internet Explorer 4.0. By enabling authors to dynamically change the rendering and content of a Web page as the user interacts with it, DHTML enables authors to create visually compelling Web sites without the overhead of server-side programs or complicated sets of controls to achieve special effects.

With DHTML, you can easily add effects to your pages that previously were difficult to achieve. For example, you can:

- Hide content until a given time elapses or the user interacts with the page.
- Animate text and images in your document, independently moving each element from any starting point to any ending point, following a predetermined path or one chosen by the user.
- Embed a ticker that automatically refreshes its content with the latest news, stock quotes, or other data.
- Use a **form** to capture user input, and then instantly process and respond to that data.

DHTML achieves these effects by modifying the in-memory representation of the current document and automatically reformatting it to show changes. It does not reload the document, load a new document, or require a distant server to generate new content. Instead, it uses the user's computer to calculate and carry out changes. This means a user does not wait for text and data to complete time-consuming round trips to and from a server before seeing the results. Furthermore, DHTML does not require additional support from applications or embedded controls to make changes. Typically, DHTML documents are self-contained, using styles and a script to process user input and directly manipulate the HTML elements, attributes, styles, and text of the document.

In short, DHTML eliminates the shortcomings of static pages. You can create innovative Web sites, on the Internet or on an intranet, without having to sacrifice performance for interactivity. Not only does DHTML enhance the user's perception of your documents, it also improves server performance by reducing requests to the server.

UNIT-IV

Introduction to WYSIWYG Design Tools

What is a WYSIWYG Editor?

What You See is What You Get (WYSIWYG) is a variety of web developing software which writes HTML code automatically while you construct your page in a word processor-esque screen. After spending several hours struggling with monotonous HTML code, this alternative sounds fantastic. There are a number of programs out there who will perform these functions. Some of the names of WYSIWYG editors include Macromedia Dreamweaver MX, Microsoft Front Page, Adobe GoLive, Netscape Composer.

Each WYSIWYG editor is completely different, and provides a unique set of tools. I prefer to use a WYSIWYG editor which allows you to have a side-by-side HTML *play-by-play*. When creating web pages, a web developer can view and change HTML code easily. Which editor you use is up to personal preference. **In class, we will use Macromedia Dreamweaver MX.**

What does the WYSIWYG do, and What Do I Need to Do?

The machines haven't taken over completely yet, humans are still needed to do most of the work when creating web pages— even with a WYSIWYG Editor. Outlined below are some of the tasks which are **not** automated.

Organizing a directory structure

This includes organizing all the files contained within a web site into folders. A single web page can contain ten or more files. The files add up quickly once your web site starts expanding. There will be more discussion about the art of organization in a future lesson.

Layout and Design

When using a WYSIWYG Editor, we are given a *blank canvas* with the tools to fill it with appealing information, colors and images. Sometimes, WYSIWYG Editors will have templates and wizards to automate layout and information management. Layout and Design will be discussed in a future lesson.

Searching for Content

No software is going to develop content for your WebPages. You, the web developer, will have to write the paragraphs, select images, and link to other pages. Tons of free clipart is available online for download, and a review on how to download images from a web page can be offered by the lesson, You can search for specific images using a search engine, or have your ordinary 35 mm film developed to disk.

Publishing to the Web

Although some website editing programs may be equipped with FTP (file transfer protocol) capabilities, it is still up to the web developer to find a website host and upload the files to make them available to the public. There will be more discussion on publishing to the web in future lessons.

Introduction to Dreamweaver

About Dreamweaver

Adobe Dreamweaver is a powerful software program used to design, create, and update non-WordPress websites. It is required to create or make non-content-related updates (e.g., navigation, layout) to a non-WordPress website. Dreamweaver uses FTP to connect and upload files to your non-WordPress website.

Usage

Dreamweaver is typically used with Adobe Contribute to setup and maintain a non-WordPress website:

1. The website is created using Dreamweaver.
2. Content updates are made using Contribute.
3. Changes to layout, navigation, and appearance are made using Dreamweaver.

Although Dreamweaver is a powerful web design program, it has a steep learning curve and high license cost. In addition, all Contribute and Dreamweaver users for a single website must use the same software version (e.g., CS3 or CS4) to avoid compatibility issues. As a result, BU departments are increasingly migrating their Dreamweaver/Contribute-based websites to WordPress.

Comparison with WordPress

WordPress has many advantages over Dreamweaver, including:

1. It’s free — no software purchase is required.
2. Ease of use.

3. Drag-and-drop navigation.
4. Streamlined file management.
5. Built-in user management.
6. Access controls.
7. A catalog of existing designs.
8. Built-in news and calendar tools.

Website Creation and Maintenance

Design

Colorado Hi-Tech Solutions can provide you with a professional and unique design that compliments your business and your field. Your current corporate branded image is taken into account along with any printed material, such as brochures and business cards, to provide a consistent look & feel in all forms of advertising media.

If you don't have any corporate branding or style already, that's no problem. We can come up with a design for your company that will be sharp and professional.

Having freedom to design something for your company is a pleasure, but we also encourage you to give us ideas and input on what you are looking for in a website. We will listen to what you are looking for and we can find ways to achieve your goals.

Development

There is a lot more that goes into building a website than just nice pictures. There are several different pieces that go into making the site show up at all.

In the frontend you have what is displayed in your browser which is made up of a mix of markup languages including: HTML, DHTML, XHTML, XML, and CSS. It also might include some other tools like a Flash element or some JavaScript for some processing or animations rather than just static text and pictures.

On the backend side or server side, there are other programming languages like PHP, Perl, ASP, and ColdFusion which can take input from forms and links and then process that information. Databases like MySQL, POSTgre, MSSQL, and others store the data in a way that is easily cross referenced or searched.

CHTS uses a combination of XHTML, CSS, JavaScript, PHP, and MySQL to develop a website that is both pleasing to look at and functional.

Content Management System

Using PHP and MySQL, CHTS has developed a CMS (content management system) that can be used to manage the content in a simple way. The CMS allows you to make changes on your website without having to install special software on your computer and without you needing to know any website languages. The CMS allows for basic editing and manipulation of text, images, and files.

Maintenance

In addition to creating new websites, CHTS can help you with your existing site. We can help you make changes to the content, add forms, or even payment options for your customers. We can help you freshen up your site's images and presentation and also get your site up to code with the new standards including getting your site to appear the same in all of the main internet browsers like Internet Explorer, Firefox, Safari, and Opera.

Design and build your website.

Create a great looking website or transfer your existing design to our content management system. We can even custom design a site for you!

Easily edit your content.

No technical knowledge required! Anyone can use our content management system to create or change their web pages.

Email notifications and reminders.

Sign up for our Community package to get basic email reminders, birthday notices and even newsletters!

Search engine optimization.

Our sites are built from the ground up with SEO in mind. Improve your ranking and attract new customers.

Web Site Hosting

Unless you have your own server, we think it makes sense to host your Freedom web site with us. As one of our valued Freedom customers, you get the benefit of our commitment to keeping your Freedom web site up and running. Our rates are competitive, our customer service is exemplary and we offer all the features you need, with room to upgrade when you need it. Our automated data backup lets you rest assured that, should there ever be a problem with our system, we can restore your site quickly and accurately.

Service Detail:

- 10GB monthly bandwidth
- 100 MB of disk storage
- 10 email addresses
- Web-based email account
- Every 3 days a full automated data backup

Additional Hosting Services:

- Domain Name + $25 per domain per year (includes yearly renewal)
- Additional Domain Names for same web site (Aliasing) + $1 per month
- SQL Database + $10 per month
- Secure Certificate for secure online transfers + $199 per year
- Dedicated IP + $20 per month (needed for secure transactions)

Understand key web concepts and terms.

- Know about the basic principles of HTML and use common HTML markup tags to modify the layout of a web page.
- Use a web authoring application to design and format web pages, format text, and work with hyperlinks and tables.
- Recognise and use common web image formats and create forms in a web page.

- Understand and use cascading style sheets.

- Prepare web pages for publishing to a web server.

Publishing Concepts

With Web publishing rules, you can allow or deny requests based on defined access policies. You can restrict access to specified users, computers, or networks, require user authentication, and inspect the traffic. Content caching enables Forefront TMG to cache Web content and to respond to user requests from the cache without forwarding the requests downstream to the published Web server. This type of content caching is called *reverse caching*. Web publishing rules have many features that determine how client Web requests are passed to the published Web servers, including the following:

- Mapping requests to specific internal paths to limit the portions of your Web servers that can be accessed.
- Delegation of user credentials for authenticating Forefront TMG to the Web server after authentication by Forefront TMG, without requiring users to supply their credentials for a second time.
- Link translation for replacing internal host names and paths in Web content with public names and external paths.
- Secure Sockets Layer (SSL) bridging, which enables Forefront TMG to inspect incoming HTTPS requests and then forward them to the Web server over an encrypted SSL channel.
- Load balancing of client requests among the Web servers in a server farm, with maintenance of client affinity for increased availability and improved performance.
- When you create a Web publishing rule in Microsoft Forefront Threat Management Gateway, you must specify a Web listener that will be associated with the rule. Each Web publishing rule is applied only to Web requests that are sent to an IP address and port specified in the Web listener associated with the rule. When a Web request matches a Web publishing rule and its associated Web listener, the properties of the Web listener determine, in particular, how Forefront TMG authenticates the user who sent the request.

XML: Introduction to XML – Markup languages

Introduction to XML

This section covers the basics of XML. The goal is to give you just enough information to get started, so you understand what XML is all about. We then outline the major features that make XML great for information storage and interchange, and give you a general idea of how XML can be used. This section of the tutorial covers:

What Is XML?

XML is a text-based markup language that is fast becoming the standard for data interchange on the Web. As with HTML, you identify data using tags (identifiers enclosed in angle brackets, like this: <...>). Collectively, the tags are known as "markup".

But unlike HTML, XML tags *identify* the data, rather than specifying how to display it. Where an HTML tag says something like "display this data in bold font" (<b>...</b>), an XML tag acts like a field name in your program. It puts a label on a piece of data that identifies it (for example: <message>...</message>).

Here is an example of some XML data you might use for a messaging application:

```
<message>
   <to>you@yourAddress.com</to>
   <from>me@myAddress.com</from>
   <subject>XML Is Really Cool</subject>
   <text>
     How many ways is XML cool? Let me count the ways...
   </text>
</message>
```

Tags and Attributes

Tags can also contain attributes -- additional information included as part of the tag itself, within the tag's angle brackets. The following example shows an email message structure that uses attributes for the "to", "from", and "subject" fields:

```
<message to="you@yourAddress.com" from="me@myAddress.com"
       Subject="XML Is Really Cool">
   <text>
     How many ways is XML cool? Let me count the ways...
   </text>
</message>
```

As in HTML, the attribute name is followed by an equal sign and the attribute value, and multiple attributes are separated by spaces. Unlike HTML, however, in XML commas between attributes are not ignored -- if present, they generate an error.

Since you could design a data structure like <message> equally well using either attributes or tags, it can take a considerable amount of thought to figure out which design is best for your purposes.

Empty Tags

One really big difference between XML and HTML is that an XML document is always constrained to be well formed. There are several rules that determine when a document is well-formed, but one of the most important is that every tag has a closing tag. So, in XML, the </to> tag is not optional. The <to> element is never terminated by any tag other than </to>.

Sometimes, though, it makes sense to have a tag that stands by itself. For example, you might want to add a "flag" tag that marks message as important. A tag like that doesn't enclose any content, so it's known as an "empty tag". You can create an empty tag by ending it with /> instead of >. For example, the following message contains such a tag:

```
<message to="you@yourAddress.com" from="me@myAddress.com"
        subject="XML Is Really Cool">
   <flag/>
   <text>
     How many ways is XML cool? Let me count the ways...
   </text>
</message>
```

Comments in XML Files

XML comments look just like HTML comments:

```
<message to="you@yourAddress.com" from="me@myAddress.com"
        subject="XML Is Really Cool">
   <! -- This is a comment -->
   <text>
     How many ways is XML cool? Let me count the ways...
   </text>
</message>
```

The XML Prolog

To complete this journeyman's introduction to XML, note that an XML file always starts with a prolog. The minimal prolog contains a declaration that identifies the document as an XML document, like this:

```
<? Xml version="1.0"?>
```

The declaration may also contain additional information, like this:

```
<? xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
```

The XML declaration is essentially the same as the HTML header, <html>, except that it uses <?..?> and it may contain the following attributes:

version

Identifies the version of the XML markup language used in the data. This attribute is not optional.

encoding

Identifies the character set used to encode the data. "ISO-8859-1" is "Latin-1" the Western European and English language character set. (The default is compressed Unicode: UTF-8.)

standalone

Tells whether or not this document references an external entity or an external data type specification (see below). If there are no external references, then "yes" is appropriate

The prolog can also contain definitions of entities (items that are inserted when you reference them from within the document) and specifications that tell which tags are valid in the document, both declared in a Document Type Definition (DTD) that can be defined directly within the prolog, as well as with pointers to external specification files. But those are the subject of later tutorials.

Why Is XML Important?

There are a number of reasons for XML's surging acceptance. This section lists a few of the most prominent.

Plain Text

Since XML is not a binary format, you can create and edit files with anything from a standard text editor to a visual development environment. That makes it easy to debug your programs, and makes it useful for storing small amounts of data. At the other end of the spectrum, an XML front end to a database makes it possible to efficiently store large amounts of XML data as well. So XML provides scalability for anything from small configuration files to a company-wide data repository.

Data Identification

XML tells you what kind of data you have, not how to display it. Because the markup tags identify the information and break up the data into parts, an email program can process it, a search program can look for messages sent to particular people, and an address book can extract the address information from the rest of the message. In short, because the different parts of the information have been identified, they can be used in different ways by different applications.

Stylability

When display is important, the stylesheet standard, XSL, lets you dictate how to portray the data. For example, the stylesheet for:

<to>you@yourAddress.com</to>
can say:

1. Start a new line.
2. Display "To:" in bold, followed by a space
3. Display the destination data.

Which produces?

To: you@yourAddress
Of course, you could have done the same thing in HTML, but you wouldn't be able to process the data with search programs and address-extraction programs and the like. More importantly, since XML is inherently style-free, you can use a completely different stylesheet to produce output in postscript, TEX, PDF, or some new format that hasn't even been invented yet. That flexibility amounts to what one author described as "future-proofing" your information. The XML documents you author today can be used in future document-delivery systems that haven't even been imagined yet.

Inline Reusability

One of the nicer aspects of XML documents is that they can be composed from separate entities. You can do that with HTML, but only by linking to other documents. Unlike HTML, XML entities can be included "in line" in a document. The included sections look like a normal part of the document -- you can search the whole document at one time or download it in one piece. That lets you modularize your documents without resorting to links. You can single-source a section so that an edit to it is reflected everywhere the section is used, and yet a document composed from such pieces looks for all the world like a one-piece document.

Linkability

Thanks to HTML, the ability to define links between documents is now regarded as a necessity. The next section of this tutorial, XML and Related Specs, discusses the link-specification initiative. This initiative lets you define two-way links, multiple-target links, "expanding" links (where clicking a link causes the targeted information to appear inline), and links between two existing documents that are defined in a third.

Easily Processed

As mentioned earlier, regular and consistent notation makes it easier to build a program to process XML data. For example, in HTML a <dt> tag can be delimited by </dt>, another <dt>, <dd>, or </dl>. That makes for some difficult programming. But in XML, the <dt> tag must always have a </dt> terminator, or else it will be defined as a <dt/> tag. That restriction is a critical part of the constraints that make an XML document well-formed. (Otherwise, the XML parser won't be able to read the data.) And since XML is a vendor-neutral standard, you can choose among several XML parsers, any one of which takes the work out of processing XML data.

Hierarchical

Finally, XML documents benefit from their hierarchical structure. Hierarchical document structures are, in general, faster to access because you can drill down to the part you need, like stepping through a table of contents. They are also easier to rearrange, because each piece is delimited. In a document, for example, you could move a heading to a new location and drag everything under it along with the heading, instead of having to page down to make a selection, cut, and then paste the selection into a new location.

How Can You Use XML?

There are several basic ways to make use of XML:

- Traditional data processing, where XML encodes the data for a program to process
- Document-driven programming, where XML documents are containers that build interfaces and applications from existing components
- Archiving -- the foundation for document-driven programming, where the customized version of a component is saved (archived) so it can be used later
- Binding, where the DTD or schema that defines an XML data structure is used to automatically generate a significant portion of the application that will eventually process that data

Traditional Data Processing

XML is fast becoming the data representation of choice for the Web. It's terrific when used in conjunction with network-centric Java-platform programs that send and retrieve information. So a client/server application, for example, could transmit XML-encoded data back and forth between the client and the server.

Document-Driven Programming (DDP)

The newest approach to using XML is to construct a document that describes how an application page should look. The document, rather than simply being displayed, consists of references to user interface components and business-logic components that are "hooked together" to create an application on the fly.

Binding

Once you have defined the structure of XML data using either a DTD or the one of the schema standards, a large part of the processing you need to do has already been defined. For example, if the schema says that the text data in a <date> element must follow one of the recognized date formats, then one aspect of the validation criteria for the data has been defined -- it only remains to write the code. Although a DTD specification cannot go the same level of detail, a DTD (like a schema) provides a grammar that tells which data structures can occur, in what sequences. That specification tells you how to write the high-level code that processes the data elements.

Archiving

The Holy Grail of programming is the construction of reusable, modular components. Ideally, you'd like to take them off the shelf, customize them, and plug them together to construct an application, with a bare minimum of additional coding and additional compilation.

The basic mechanism for saving information is called *archiving*. You archive a component by writing it to an output stream in a form that you can reuse later. You can then read it in and instantiate it using its saved parameters. (For example, if you saved a table component, its parameters might be the number of rows and columns to display.) Archived components can also be shuffled around the Web and used in a variety of ways.

When components are archived in binary form, however, there are some limitations on the kinds of changes you can make to the underlying classes if you want to retain compatibility with previously saved versions.

SAX

Simple API for XML

This API was actually a product of collaboration on the XML-DEV mailing list, rather than a product of the W3C. It's included here because it has the same "final" characteristics as a W3C recommendation.

You can also think of this standard as the "serial access" protocol for XML. This is the fast-to-execute mechanism you would use to read and write XML data in a server, for example. This is also called an event-driven protocol, because the technique is to register your handler with a SAX parser, after which the parser invokes your callback methods whenever it sees a new XML tag.

DOM

Document Object Model

The Document Object Model protocol converts an XML document into a collection of objects in your program. You can then manipulate the object model in any way that makes sense. This mechanism is also known as the "random access" protocol, because you can visit any part of the data at any time. You can then modify the data, remove it, or insert new data.

DTD

Document Type Definition

The DTD specification is actually part of the XML specification, rather than a separate entity. On the other hand, it is optional -- you can write an XML document without it. And there are a number of schema proposals that offer more flexible alternatives. So it is treated here as though it were a separate specification.

Namespaces

The namespace standard lets you write an XML document that uses two or more sets of XML tags in modular fashion. Suppose for example that you created an XML-based parts list that uses XML descriptions of parts supplied by other manufacturers.

XSL(Extensible Stylesheet Language)

The XML standard specifies how to identify data, not how to display it. HTML, on the other hand, told how things should be displayed without identifying what they were. The XSL standard has two parts, XSLT (the transformation standard, described next) and XSL-FO (the part that covers *formatting objects*, also known as *flow objects*). XSL-FO gives you the ability to define multiple areas on a page and then link them together. When a text stream is directed at the collection, it fills the first area and then "flows" into the second when the first area is filled. Such objects are used by newsletters, catalogs, and periodical publications.

.XSLT(+XPATH)
Extensible Stylesheet Language for Transformations

The XSLT transformation standard is essentially a translation mechanism that lets you specify what to convert an XML tag into so that it can be displayed -- for example, in HTML. Different XSL formats can then be used to display the same data in different ways, for different uses. (The XPATH standard is an addressing mechanism that you use when constructing transformation instructions, in order to specify the parts of the XML structure you want to transform.)

XML Schema

A large, complex standard that has two parts. One part specifies structure relationships. (This is the largest and most complex part.) The other part specifies mechanisms for validating the content of XML elements by specifying a (potentially very sophisticated) *data type* for each element. The good news is that XML Schema for Structures lets you specify any kind of relationship you can conceive of. The bad news is that it takes a lot of work to implement, and it takes a bit of learning to use. Most of the alternatives provide for simpler structure definitions, while incorporating the XML Schema data type standard.

RELAX

Regular Language description for XML

Simpler than XML Structure Schema, RELAX uses XML syntax to express the structure relationships that are present in a DTD, and adds the XML Data type Schema mechanisms, as well. Includes a DTD to RELAX converter.

SOX

Schema for Object-oriented XML

SOX are a schema proposal that includes extensible data types, namespaces, and embedded documentation.

TREX

Tree Regular Expressions for XM

A means of expressing validation criteria by describing a *pattern* for the structure and content of an XML document. Includes a RELAX to TREX converter.

Schema Linking and Presentation Standards

Arguably the two greatest benefits provided by HTML were the ability to link between documents, and the ability to create simple formatted documents (and, eventually, very complex formatted documents). The following standards aim at preserving the benefits of HTML in the XML arena, and to adding additional functionality, as well.

XML Linking

These specifications provide a variety of powerful linking mechanisms, and are sure to have a big impact on how XML documents are used.

XLink: The XLink protocol is a proposed specification to handle links between XML documents. This specification allows for some pretty sophisticated linking, including two-way links, links to multiple documents, "expanding" links that insert the linked information into your document rather than replacing your document with a new page, links between two documents that are created in a third, independent document, and indirect links (so you can point to an "address book" rather than directly to the target document -- updating the address book then automatically changes any links that use it).

XML Base: This standard defines an attribute for XML documents that defines a "base" address, that is used when evaluating a relative address specified in the document. (So, for example, a simple file name would be found in the base-address directory.)

XPointer: In general, the XLink specification targets a document or document-segment using its ID. The XPointer specification defines mechanisms for "addressing into the internal structures of XML documents", without requiring the author of the document to have defined an ID for that segment. To quote the spec, it provides for "reference to elements, character strings, and other parts of XML documents, whether or not they bear an explicit ID attribute".

Features of Mark up languages

A **markup language** is a modern system for annotating a document in a way that is syntactically distinguishable from the text. The idea and terminology evolved from the "*marking up*" of manuscripts, i.e., the revision instructions by editors, traditionally written with a blue pencil on authors' manuscripts. Examples are typesetting instructions such as those found in troff, TeX and LaTeX, or structural markers such as XML tags. Markup instructs the software displaying the text to carry out appropriate actions, but is omitted from the version of the text that is displayed to users. Some markup languages, such as HTML, have pre-defined presentation semantics, meaning that their specification prescribes how the structured data are to be presented; others, such as XML, do not.

Presentational markup

> The kind of markup used by traditional word-processing systems: binary codes embedded in document text that produces the WYSIWYG effect. Such markup is usually designed to be hidden from human users, even those who are authors or editors.

Procedural markup

> Markup is embedded in text and provides instructions for programs that are to process the text. Well-known examples include troff, LaTeX, and PostScript. It is expected that the processor will run through the text from beginning to end, following the instructions as encountered. Text with such markup is often edited with the markup visible and directly manipulated by the author. Popular procedural-markup systems usually include programming constructs, so macros or subroutines can be defined and invoked by name.

Descriptive markup

> Markup is used to label parts of the document rather than to provide specific instructions as to how they should be processed. The objective is to decouple the inherent structure of the document from any particular treatment or rendition of it. Such markup is often described as "semantic". An example of descriptive markup would be HTML's **<cite>** tag, which is used to label a citation.

There is considerable blurring of the lines between the types of markup. In modern word-processing systems, presentational markup is often saved in descriptive-markup-oriented systems such as XML, and then processed procedurally by implementations. The programming constructs in descriptive-markup systems such as TeX may be used to create higher-level markup systems which are more descriptive, such as LaTeX.

Features

A common feature of many markup languages is that they intermix the text of a document with markup instructions in the same data stream or file. This is not necessary; it is possible to isolate markup from text content, using pointers, offsets, IDs, or other methods to co-ordinate the two. Such "standoff markup" is typical for the internal representations that programs use to work with marked-up documents. However, embedded or "inline" markup is much more common elsewhere. Here, for example, is a small section of text marked up in HTML:

```
<h1> Anatidae </h1>
<p>
The family <i>Anatidae</i> includes ducks, geese, and swans,
but <em>not</em> the closely related screamers.
</p>
```

XML naming rules

Common Naming Rules

Unique Elements and Types Naming Rule

Whenever a new element or type is declared, it should be named uniquely within its schema, name spaces and across all name spaces.

This structure follows the ebXML recommendations for naming, which are base upon the ISO 11179 standard "Information technology — Specification and standardization of data elements". This naming convention is backed by a broad international consensus.

The object term in the ObjectPropertyRepresentation scheme describes the data object represented by the element and its type. If the element and its type occur in the context of an object or the object is unknown, the object term MAY be omitted.

The table below shows examples of names under this naming scheme:

Object	Property	Representation	Element name
Communication	Mode	Code	CommunicationModeCode
Country		Code	CountryCode
Currency	Exchange	Rate	CurrencyExchangeRate
Location	Type	Code	LocationTypeCode
Transport	Method	Code	TransportMethodCode
Street	Name		StreetName

OrganizationRegistration		Date	OrganizationRegistrationDate
Address	Type	Code	AddressTypeCode

Singular Form of Names

For example, “Premises” is a plural name so it is used as is in “PremisesArabicName”.

Connecting Words

Since we are not using full sentences as element names, connecting words SHALL NOT be used in the element names.

Characters Permitted in Names

Alphanumeric characters are A to Z, a to z, and 0 to 9.

Language Options

Arabic /English Naming Options

In general, English is language usage is recommended for naming elements, types, and attributes. In case English language is inconvenient, Arabic language may be used.

XML Schema Language

For consistency reasons mixed language SHALL NOT be used in one XML schema. So the whole XML schema SHALL be in either Arabic or English language.

Declaring XML Schema Language

“xml:lang” root element attribute SHALL be given “ar” value in case XML schema elements names are in, and “en” SHALL be given to the attribute in case XML schema elements names are in English

XML Schema Language Verification

Naming in a specific language SHOULD follow the syntactic and orthographical rules of that language, as specified in the relevant dictionaries and specialist literature.

XML Schema Cross References

In order to secure consistency between XML schemas, an XML schema SHALL NOT refer to (include or import) schemas in different language.

Reusing Types from Different Languages

In order to secure consistency between XML schemas, an XML schema SHALL NOT reuse a type (simple or complex) defined in a different language.

Inheriting Types from Different Languages

In order to secure consistency between XML schemas, an XML schema SHALL NOT iherit a type (simple or complex) defined in a different language.

Types (Simple and Complex) Naming

Simple Type Suffix

The names of simple data types SHALL end with the text string "Type". In the element name, this suffix is omitted. This way it is always possible to distinguish types from elements.

Complex Type Suffix

The names of complex data types SHALL end with the text string "Structure". In the element name, this suffix is omitted. This way it is always possible to distinguish types from elements.

UpperCamelCase Use in Types Naming

UpperCamelCase means that the name SHALL begin with an upper case letter, and the following new words SHALL begin with upper case letters as well, example:

<element name="StreetBuildingIdentifier" type="cc:StreetBuildingIdentifierType">

Elements Naming

The Relation Between Element Names and Type Names

The name of the element SHOULD directly reflect the name of the type used. This is done by deleting the “Type”/”Structure” suffix from the type name. This rule only applies to elements that have their type defined in the same schema.

UpperCamelCase Use in Elements Naming

UpperCamelCase means that the name SHALL begin with an upper case letter, and the following new words SHALL begin with upper case letters as well, example:

```
<xsd:element name="StreetBuildingIdentifier" type="cc:StreetBuildingIdentifierType"/>
```

Attributes Naming

lowerCamelCase Use

lowerCamelCase begins with a lower case letter, and every following word SHOULD begin with a capital letter.

```
<xsd:complexType name="CustomerType">

        <xsd:sequence>

                <xsd:element name="GivenName" type="prefix:GivenNameType"/>

                ...

        </xsd:sequence>

        <xsd:attribute name="customerIdentifier" type="prefix:CustomerIdentifierType"/>

</xsd:complexType>
```

Naming of YEFI XML Schema and Metadata Files

Naming of A YEFI XML Schema File

It is important to follow a specific naming pattern for the schemas and metadata files. This will enable users to identify the purpose and version of the file by looking at the name.

Version format is "-vM-N", where M is the major version and N is the minor version.

Example:

PersonProfileTypes-v1-0.xsd

Naming of Metadata File

This ensures that the metadata of an element can be identified by its file name.

Building block of XML Document

Elements

Elements are the **main building blocks** of both XML and HTML documents.

Examples of HTML elements are "body" and "table". Examples of XML elements could be "note" and "message". Elements can contain text, other elements, or be empty. Examples of empty HTML elements are "hr", "br" and "img".

Examples:

```
<body>sometext</body>
```

```
<message>some text</message>
```

Attributes

Attributes provide **extra information about elements**.

Attributes are always placed inside the opening tag of an element. Attributes always come in name/value pairs. The following "img" element has additional information about a source file:

```
<img src="computer.gif" />
```

The name of the element is "img". The name of the attribute is "src". The value of the attribute is "computer.gif". Since the element itself is empty it is closed by a " /".

Entities

Some characters have a special meaning in XML, like the less than sign (<) that defines the start of an XML tag.

Most of you know the HTML entity: "&nbsp;". This "no-breaking-space" entity is used in HTML to insert an extra space in a document. Entities are expanded when a document is parsed by an XML parser.

The following entities are predefined in XML:

Entity References	**Character**
&lt;	<
&gt;	>
&amp;	&
&quot;	"
&apos;	'

PCDATA

PCDATA means parsed character data.

Think of character data as the text found between the start tag and the end tag of an XML element.

PCDATA is text that WILL be parsed by a parser. **The text will be examined by the parser for entities and markup**.

CDATA

CDATA means character data.

CDATA is text that will NOT be parsed by a parser. Tags inside the text will NOT be treated as markup and entities will not be expanded.

Difference between HTML & XML

HTML (**H**yper**T**ext **M**arkup **L**anguage)	XML (e**X**tensible **M**arkup **L**anguage)
HTML has some predefined tags with some predefined meaning.	XML has no tags.
You can use the tags as it is in HTML. You can't write new tags.	Always you have to write new tags according to your requirement.
HTML document is used to present the data to user in the required format.	XML document is used to store the data (XML document is container for the data).
HTML document may not be well formed document.	XML document must be a well-formed document.
In HTML there is no data type.	In XML some data types are defined.
EX: <html> <title>Example</title> <head>Head Part</head>	EX: <?xml version="1.0"?> <student> <name>**Raj**</name>

<body> This is HTML </body> </html>	<class>**12**</class> <roll-no>**515**</roll-no> </student>

XML Parser

This module provides ways to parse XML documents. It is built on top of XML::Parser::Expat, which is a lower level interface to James Clark's expat library. Each call to one of the parsing methods creates a new instance of XML::Parser::Expat which is then used to parse the document. Expat options may be provided when the XML::Parser object is created. These options are then passed on to the Expat object on each parse call. They can also be given as extra arguments to the parse methods, in which case they override options given at XML::Parser creation time.

The behavior of the parser is controlled either by "Style" and/or "Handlers" options, or by "set Handlers" method. These all provide mechanisms for XML::Parser to set the handlers needed by XML::Parser::Expat. If neither Style nor Handlers are specified, then parsing just checks the document for being well-formed.

When underlying handlers get called, they receive as their first parameter the *Expat* object, not the Parser object.

Expat is an event based parser. As the parser recognizes parts of the document (say the start or end tag for an XML element), then any handlers registered for that type of an event are called with suitable parameters. All handlers receive an instance of XML: Parser::Expat as their first argument. of the methods that can be called on this object.

Components of XML

Components of XML is a set of rules and declarations for an XML document markup language for a specific document type. XML Schema Definitions extend the functionality of a DTD and XDR in several ways. An XSD is a XML document and allows for a common syntax with other XML components, specification of primitive data types, and provides the ability to reference other schemas as well as namespace integration ability. In XML, a namespace can be defined to qualify element names and structure in a unique fashion. If an XML document uses a XSD, it must be applied programmatically during processing. Figure 5 presents a XML
Schema Definition that is associated with the XML document.

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

```
<xsd:schema xmlns:xsd=”http://www.w3.org/2001/XMLSchema”>
<xsd:element name=”digitalversatiledisks” type=”digitalversatiledisksType”/>
<xsd:complexType name=”digitalversatiledisksType”>
<xsd:sequence>
<xsd:element name=”title”/>
<xsd:element name=”genre”/>
<xsd:element ref=”rating” type=”MPARating”/>
<xsd:element name=”runtime”/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name=”MPARating”>
<xsd:sequence>
<xsd:element name=”PG” type=”xsd:string”/>
<xsd:element name=”PG-13” type=”xsd:string”/>
<xsd:element name=”R” type=”xsd:string”/>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

DTD’s Using XML with HTML and CSS

I have a project due tomorrow, i have basically finished it and am awaiting a reply from my lecturer on how the information is actually meant to be presented... ok below is a simple example of what is required

Create an XML template
Create a DTD
Use CSS to style

So for my DTD as an example person.dtd

```
           <!ELEMENT person (name_first,name_last,personal_details+)>
  <!ELEMENT name_first   (#PCDATA)>
  <!ELEMENT name_last    (#PCDATA)>

          <!ELEMENT personal_datails (age,weight)>
          <!ELEMENT age (#PCDATA)>
          <!ELEMENT weight (#PCDATA)>
```

Then an xml file person.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="person.css"?>
<! DOCTYPE person SYSTEM "person.dtd">

<people>
        <person>
        <name_first>bbq</name_first>
        <name_last>D.I.C</name_last>
        <personal_details>
                <age>20</age>
                <weight>80</weight>
                </personal_details>
        </person>

        <person>
        <name_first>John</name_first>
        <name_last>Smith</name_last>
                <personal_details>
                        <age>50</age>
                        <weight>80</weight>
                </personal_details>
        </person>
</people>
```

Then in my .css file called person.css I have the following

```
person
{
        font-size: 16pt;
        font-family: Arial Black;
        background-color: #dddddd;
        width: 70%;
        text-align: center;
        border : solid 2px;
}
```

The problem lies in that i have many other DTD all under person, not only do i have name i have things like personal details which in that has age, height, weight etc..

When i open the XML document it is rather horrible to look at, with information all being in. How can i use CSS to format my xml, as in like have each thing on a new line... The above is an example and it hasn't been through a validator, however yeah, i cannot find anywhere how to format my xml document using css. As far as i know i cannot use XSL either, which makes it a pain. So the main thing is, it all comes out in a single grey box, all on one line I would like to have it in the format

TEXT BOOKS

[T1] The complete reference HTML, by Thomas A Powell, TMH publication.

[T2] Mastering HTML 4.0 by Deborah S. Ray and Erich J. Ray. BPB Publication.

[T3] Internet and World Wide Web Deitel HM, Deitel, Goldberg, Third Edition

REFERNCES

[R1] HTML Black Book, Stephen Holzner, Wiley Dreamtech.

[R2]Rajkamal, “Web Technology”, Tata McGraw-Hill, 2001.

[R3] Jeffrey C. Jackson, “Web Technologies: A Computer Science Perspective”, Pearson.

[R4]XML How to Program by Deitel Deitel Nieto.